# Multithreaded Design in the Athena Environment

by

Greg Hudson

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1996



Author .....
Department of Electrical Engineering and Computer Science
February 6, 1996

Certified by .....
Jeffrey I. Schiller
Network Manager
Thesis Supervisor

Accepted by .....
F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Theses

# Multithreaded Design in the Athena Environment

by

Greg Hudson

Submitted to the Department of Electrical Engineering and Computer Science
on February 6, 1996, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

## Abstract

In this thesis, I explored the advantages and disadvantages of using concurrency within a single address space (threads) in the design of software used in the Athena environment. In addition to discussing the theoretical advantages and disadvantages of multithreaded designs as applied to the software in the Athena environment, I also discuss the practical results of three implementation projects where an existing piece of software in the Athena environment was converted to support or use multithreaded design.

Thesis Supervisor: Jeffrey I. Schiller
Title: Network Manager

# Acknowledgments

I would like to thank Christopher Provenzano for his invaluble efforts on the MIT Pthreads library and his interest and assistance in the progress of this work.

# Contents

# Chapter 1

# Introduction

Most computer programs written today use only a single flow of control; that is, they perform only one task at a time. Systems often make use of multiple processes[1] running at once on the same computer, but each program has only one flow of control and is not able (except under very special circumstances) to directory read from or write to the memory used by another program in the system.

A multithreaded program uses several flows of control, or threads, within a single process. Threads within a process operate within the same address space, so they are all able to read from and write to the same memory. Although multithreaded programming is not a new idea, it has not been popular in the Unix environment until recently. In the last few years, popular Unix operating systems such as Solaris have begun integrating support for multithreaded programming, and Unix applications have begun to appear using threads.

The purpose of this thesis is to evaluate the feasibility of using multithreaded design at the current time in projects involving the software making up MIT's Project Athena Computing Environment[3]. The three specific areas I explore are:

- What are the general advantages and disadvantages of multithreaded design, and how do they apply to software in the Athena environment?

[1]By "process" I mean an execution of a program within an address space separated from other processes. There may, of course, be multiple processes running the same program on a given computer at a given time.

- What is the current status of the tools for running and debugging multithreaded programs? What is the short-term potential for improving these tools where they fall short?

- What are the practical results of trying to apply multithreaded design to existing software in the Athena environment?

To answer the last question, I conducted three implementation projects: conversion of the Hesiod service name resolution library to support multithreaded programs, conversion of the Common Error Description library to support multithreaded programs, and conversion of the `attach` program to conduct its individual tasks in parallel.

In two appendices to this thesis, I provide the code for the implementation projects I carried out as well as information on using the tools currently available in the Athena environment for multithreaded development.

# Chapter 2

# Theoretical Considerations

This chapter discusses some of the theoretical benefits and costs of threads, and how they apply to the Athena environment. The specific areas I will discuss are performance, design simplification, design restrictions, stability, and testing.

When discussing the relative merits of multithreaded design, it is important to understand what the alternatives are. Suppose you have a system specification which could be satisfied by a design involving multiple threads performing different tasks within a single process. Each thread's task could involve at various times waiting for a certain event to occur (input arriving from some source, acquisition of a lock on some resource, a timer going off, etc.). There are three basic alternative designs which use only a single thread per process:

- The *event-loop* model. In this model, the system uses one process whose flow of control is centered around an event loop. Every time an event occurs which allows a task to be continued, the process performs as much of that task as can be done without waiting for another event. As soon as the task needs to wait for another event to occur, it registers that fact with the event loop code and returns to the event loop. The Zephyr servers in the Athena environment use the event-loop model.

  A variation on the event-loop model is the *queueing* model, wherein events are placed in queues as they arrive and processed according to some priority scheme.

This approach can improve performance over the vanilla event-loop model when some kinds of events are more important than others, or when data must be retrieved from the Unix kernel quickly to avoid overflowing kernel resources (for example, received User Datagram Protocol packets). The queueing model was used in the server for the Remote Virtual Disk protocol used at MIT Project Athena.

- The *forking* model. In this model, the system uses a process for each task to be performed. If multiple tasks ever need to modify the same state information, they either communicate with a designated master process via some interprocess communication system (either some form of message queue or shared memory and semaphores), or keep the shared state in a file and synchronize access somehow (either with locks or by tolerating loose synchronization). The popular InterNetworkNews server[6] uses the forking model to handle connections from newsreading clients.

- The *worker process* model, a variant of the event-loop model where a "master" process spawns off a fixed number of "slave" subprocesses to handle events in parallel. Each process typically uses the event-loop model to respond to events. If multiple tasks need to modify the same state information, they must do so in the same manner as the forking model[1]. Netscape's World Wide Web server is reportedly an example of a system using the worker process model.

In some cases, the designer of a system can make simplifying assumptions to come up with a simpler model than the above. As a simple example, if a system's requirements can be satisfied by a design involving only one task, no special effort is needed to implement that design. The following discussion will be targeted at systems which are most easily thought of as performing multiple tasks at any one time.

[1] In special cases, it may be possible to ensure that all such events are handled by the same slave process.

## 2.1 Performance

One of the most common reasons for using multithreaded design in programs is performance. A program using the event-loop model will only be able to use one CPU and, under most Unix systems, can make only one filesystem access at a time[2]. On a dedicated server machine with multiple CPUs, such a restriction on performance may be unacceptable. The forking model allows concurrent use of all system resources; however, the use of many processes tends to artificially load down a machine in three ways: first, the overhead of spawning a process is often noticeable both in terms of system time and kernel resources; second, Unix schedulers are often not equipped to handle large numbers of processes, leading to unnecessary delays; and third, data may have to be replicated between processes because they do not have access to each others' address space[3]. The worker process model allows current access to system resources without imposing extra load on the machine; however, the worker process model can cause extra data copies and context switches because events may have to arrive at the master process and be communicated to the slave processes.

A multithreaded design can potentially address the performance problems of all three models, since a good thread scheduler will allow concurrent access to resources by different threads, threads generally require fewer resources than processes because they do not need separate addresss spaces, and multiple threads can communicate very quickly through the memory they share. Note, however, that many of the available tools for multithreaded development currently run all threads within a single process with no kernel support, leading to the same performance limitations as the event-loop model.

Performance is not generally a convincing argument for threads in the Athena environment, which does not use many machines with multiple processors or disk controllers[4]. When high performance is desired, the event loop model typically pro-

[2]Although Unix provides non-blocking primitives for network and pipe I/O, filesystem operations on most Unix systems are always blocking.

[3]For instance, the INN news server stores a separate copy of the list of the system's newsgroups in each subprocess.

[4]Multiprocessor machines are currently in use for the mail servers, whose software uses a forking

vides close to optimum performance on a single-processor machine with one disk controller.

## 2.2 Design Simplicity

As noted above, this chapter's discussion is targeted at systems which are best thought of as performing multiple tasks at once. Not all complicated systems fall into this model; in paricular, user interfaces can become very complicated but are still best thought of as responding to user actions (i.e. events). The following comments apply best to server applications and other systems with many heretogeneous tasks.

In general, the event-loop model allows easy manipulation of shared state, but becomes very complicated when the number of different kinds of tasks increases or when the types of events they must wait for increases. Every task must be effectively decomposed into a state machine with external events as transitions, and the event loop must make transitions in the state machine of each task. As a result, the work of each task is expressed in an artifically fragmented manner, and the main loop becomes increasingly complicated as the system's tasks become more heterogeneous.

Conversely, the forking model allows the designer to express each task's work in a natural manner, but makes manipulation of shared state very difficult. Typically, manipulations of shared state must be expressed as loosely synchronized operations on copies of the shared state, or they must be designed to fit into the paradigm of messages between processes. Although modern Unix systems provide primitives for explicitly sharing memory and synchronizing processes, they are nonportable and often cumbersome to use; in addition to the overhead of identifying and attaching shared memory segments, one must generally provide one's own memory allocation package to allocate memory within a region of explicitly shared memory.

The worker process model has the complications of both the event-loop model and the forking model, making it the hardest to use. Each worker process has to follow the

model; machines with multiple disk controllesr are in use for file servers, whose software uses multiple threads.

design constraints of the event-loop model, and any state relevant to more than one worker process must conform to the design constraints of the forking model. Systems which use the worker process model tend to place restrictions on the kinds of events handled by each worker process in order to minimize the amount of inter-process state.

A multithreaded design can simultaneously allow the natural expression of multiple tasks and access to shared state, but there are two serious costs:

- All accesses to state shared between tasks must be locked[5] Locks consume space and take time to use, and it is very difficult to verify that all accesses are properly locked. Moreover, if locks are acquired in the wrong order, deadlock may result.

- Multithreaded programs cannot make efficient use of any interfaces which refer to static data. Because C does not provide garbage collection and encourages static or stack allocation rather than heap allocation, it is often difficult to provide reentrant interfaces to complicated functions. For instance, consider the Unix `gethostent()` family of functions, which return a `hostent` structure containing a variable-length name, a variable-length list of aliases (each of which is itself variable-length), an a variable-length list of addresses (each of which is itself variable-length as well). A reentrant interface to such a function must either be terribly complicated or must violate type safety by accepting a block of untyped memory in which to return the answer.

It is worth noting that even single-threaded programs run into difficulty with complicated interfaces like the `gethostent()` family of functions. Since `gethostent()` returns its result in a static buffer, a library cannot save the result of a `gethostent()` function between two library function calls; it must copy out the relevant information, and there is no easy and portable way of copying all of the information from one `hostent` structure into another. Similarly, every time a single-threaded program makes use of any library function after a `gethostent()` family routine, it must

[5]The POSIX standard provides for an atomic integral type `sig_atomic_t` to which read and write accesses are atomic, but obviously most interesting data objects are aggregate, and most interesting data accesses are of the read-modify-write flavor.

worry whether the library function will call any `gethostent()` routine which might overwrite the static buffer[6]. Nevertheless, all of the problems encountered in single-threaded programs are exacerbated in a multithreaded program which cannot easily control the synchronization of calls to functions returning data in static buffers.

There is a design strategy, sometimes called *coroutines*, in which flows of control only yield at explicit points or when they call a blocking operation. Coroutines partially circumvent the design restrictions of multithreaded programs, but they cannot take advantage of multiple processors and they still require programmers to worry about the use of shared data across calls to library functions that might block.

The design advantages of threads apply to a few Athena subsystems, but not most. The vast majority of client programs are best understood as performing only a single task at a time and do not encounter any design problems as single-threaded programs. Since the Kerberos protocol is stateless and packet-oriented, the Kerberos server poses no difficulty for an event-loop design. Multithreaded design could be applied fruitfully to the Zephyr server (the code for which is currently dominated by packet retransmission logic), to the INN news server, and to other servers which use both heterogeneous tasks and global state.

## 2.3 Stability and Testing

A common complaint about multithreaded system design is that the number of paths through the program grows very quickly, making it impossible to approximate path-complete testing. This opens up the possibility of subtle bugs occurring during uncommon executions of the program. Such bugs are called "race conditions" because they only occur during specific executions where one thread "races" against another thread for access to a resource. Race conditions can be difficult to reproduce and thus difficult to track down. Although unpredictable bugs are nothing new to designers of complicated systems, this is not a criticism to be taken lightly. Indeed, the only good

[6] As an example, almost every mail-related program in the Athena environment has fallen prey to the problem of passing a hostname returned by `gethostbyname()` to the `krb_getrealmofhost()` function, which itself uses `gethostbyname()`.

approach to dealing with race conditions in multithreaded programs is prevention. It is usually sufficient to wrap all accesses to all shared aggregate data types inside functions which acquire a lock upon entry and release it upon exit. Note that the forking model also produces a very large number of executions, although the strong memory protection afforded by the forking model reduces the potential for race conditions.

Another category of subtle and unpredictable bugs in C programs is memory corruption errors, whereby an access to a dangling pointer (that is, a pointer to memory which has accidentally been deallocated) or out-of-bounds access to an array corrupts memory used elsewhere in the program. Relative to an event-loop design, a multithreaded design can reduce the likelihood of memory corruption errors by restricting the logical lifetime of data objects. If an object is only accessed within a single procedure, it can be stack-allocated and used without fear of dangling pointers. If that procedure had to be split up because of the constraints of the event loop model, the object will have to be heap-allocated and set aside to be referenced later. Memory leaks are similarly less likely in a multithreaded program which can take advantage of stack storage than in a program using an event-loop design.

Thus, relative to the forking model, a multithreaded design sacrifices memory protection; relative to the event-loop model, a multithreaded design sacrifices determinism but may improve stability by reducing object lifetime. It is difficult to determine a priori how these considerations are applicable to the Athena environment. I will describe my own experiences during my discussion of the implementation projects.

# Chapter 3

# Tools

This chapter describes some of the available tools for multithreaded development, their current usability, and their short-term potential for improvement. There are many multithreaded environments and tools I will not discuss here, since they are either not applicable to or have not been used in the Athena environment.

In discussing the tools below, I will refer to the "POSIX standard"; this phrase refers to the IEEE 1003.1c standard (POSIX.1c for short), approved as an IEEE standard in June of 1995. See [2] or [4] for a description of the POSIX threads standard. Here are some brief notes about the POSIX.1c standard:

- As one would expect, POSIX.1c specifies interfaces for creating threads, locking against other threads (mutexes), signalling other threads (condition variables), and waiting for a thread to complete.

- Unlike some other thread interfaces, POSIX.1c specifies interfaces for cancelling threads. Threads can allow themselves to be terminated either synchronously (at explicit "cancellation points" during the execution of the cancelled thread) or asynchronously, although asynchronous cancellation requires very careful precautions.

- POSIX.1c allows the application to create a thread as either "globally scheduled" or "process-scheduled," which roughly translates into "kernel thread"

and "user-space thread." Globally scheduled threads usually allow more concurrency, while process-scheduled threads do not consume kernel resources. Of course, a particular implementation of POSIX.1c may provide only one kind of thread or the other.

- Unlike some other thread interfaces, POSIX.1c does not specify interfaces for suspending and resuming threads, for acquiring a global lock on all thread execution, for waiting for one of a specified set of threads to complete, or for waiting for one of a specified set of condition variables to be signalled.

- POSIX.1c specifies reentrant interfaces for C library functions which previously used static data for return values or internal state. For instance, the function `strtok()` uses internal state to keep track of the string being tokenized; POSIX.1c specifies `strtok_r()` as a reentrant interface where the caller passes in a pointer to the space which should be used for internal state.

  Note that some important areas of system functionality such as networking code are not covered by the POSIX standards, so POSIX.1c does not provide reentrant interfaces to those routines.

## 3.1 MIT Pthreads

There are several free software libraries available which implement all or part of the POSIX.1c standard. The only one installed and used in the Athena environment is, unsurprisingly, one developed at MIT by Christopher Provenzano and commonly called "MIT Pthreads". MIT Pthreads has the distinction of supporting ten platforms, including Ultrix, Solaris, IRIX, Linux, and NetBSD (but not AIX). MIT Pthreads also attempts to implement all of the POSIX standard by providing its own version of large parts of the C library (other thread libraries claim to implement the POSIX standard but ignore the required library routines). Moreover, as a concession to existing code, routines such as `strtok()` which don't have reentrant interfaces use thread-specific data instead of static buffers so that they may be used

simultaneously in multiple threads. Finally, MIT Pthreads includes the only currently available thread-safe Domain Name Service resolver other than the Solaris resolver.

On the other hand, MIT Pthreads has the following weaknesses:

- It is a user-level threads package, and therefore will not allow concurrent use of multiple CPUs or disks by a single process.
- Because the Unix `select()` mechanism only allows a process to wait for a file descriptor event, signal, or timeout, any other events such as waiting for a lock on a file must be polled for by the threads library, with corresponding performance problems.
- Because it provides most of a C library, it may replace a C library routine with one that doesn't interact properly with system configuration files. (Ideally the threads library knows how to interact properly with the system configuration files on each system, of course.)
- MIT Pthreads will not interrupt a system call in order to run a signal handler; thus, if all of a process's threads are in the middle of locks or system calls and the process receives a signal, the signal will not be delivered until some thread wakes up naturally.
- MIT Pthreads tends to interact poorly with libraries not compiled against the MIT Pthreads header files, for several reasons: the library may use the operating system's `getc()` and `putc()` macros which are generally incompatible with the MIT Pthreads standard I/O implementation; the library may use the global constant `errno` and therefore not be able to detect errors properly; or the library may contain inlined system calls (Pthreads relies on being able to intercept system calls).
- A few of the reentrant C library routines are not yet implemented.

Because the source code for Pthreads is available and freely redistributable, it is possible to work around or fix some of these problems. Future releases of Pthreads are

expected to allow the use of kernel-level threads where the operating system supports it, and to implement the remaining C library functions specified by POSIX.1c.

MIT Pthreads is probably the most useful tool for running multithreaded programs on systems whose operating systems don't support kernel threads. As a user-space threads library with freely available source, MIT Pthreads also provides some debugging opportunities not available in native threads environments, as I will discuss shortly. Nevertheless, it is often less painful to use a multithreaded environment integrated into the operating system; I will discuss such an operating system below.

## 3.2 Solaris Threads

One of the most widely used native threads environments is the environment in Sun's Solaris operating system. Solaris version 2.3 (the version currently used in the Athena environment) and version 2.4 do not implement the POSIX threads standard, but the interface Solaris does implement is functionally very similar to the POSIX threads standard[1], since Solaris threads were one of the important models for the POSIX threads standard. Solaris version 2.5 contains a minimal implementation of the POSIX threads standard.

As part of the operating system, the Solaris thread facilities do not have many of the problems that MIT Pthreads does. Solaris implements a hybrid threads system; that is, it supports both kernel-level and user-level threads, so the application can have concurrent access to multiple CPUs and disks. The Solaris thread facilities are not prone to improper use of system configuration files as MIT Pthreads is. Unfortunately, Solaris does not solve the problem of interaction between libraries and multithreaded applications; all code intended for a multithreaded application must be compiled with `-D_REENTRANT`, while code compiled for a singlethreaded application must not be[2].

[1]The primary difference is that the Solaris threads interface does not provide support for cancellation of threads, and does provide support for suspending and resuming threads.

[2]There are two reasons for this: first, older versions of Solaris contain a shared C library which won't operate properly with multithreaded code, and second, `getc()` and `putc()` are required to

In a Usenet discussion in `comp.programming.threads`, Peter Eriksson and Simon E. Spero pointed out some notable bugs in the threads facilities of Solaris 2.4:

- The YP and NIS+ name lookup modules contain deadlocking problems.
- `accept()` is not thread-safe when called with the same file descriptor concurrently.
- The `fork()` library routine calls `malloc()` in the child process; if `malloc()` was locked by another thread in the parent, the child process will hang.

Of course, the most important limitation of the Solaris threads facilities is that they only work on Solaris, and are thus only a partial solution for programs which must run on multiple platforms. To avoid being tied to Solaris, applications which wish to use the Solaris thread facilities should use the POSIX threads interface and use macros to translate POSIX calls to Solaris calls until Solaris 2.5 is in use in the Athena environment. See Appendix B for an example header file which performs translations for the important parts of the POSIX interface.

## 3.3 DCE Threads

Part of the Open Software Foundation's Distributed Computing Environment is a user-space threads library. As a commercially supported, cross-platform library, the DCE threads system at first seems like it might be a useful solution for compiling threaded programs on platforms without native thread support. In addition to POSIX.1c draft 4 functionality, the DCE threads system supplies an exception-handling system and primitives for acquiring a global lock on thread execution. Unfortunately, there are several reasons why the DCE threads system is probably unsuitable for use in the Athena environment:

---

be thread-safe under both the Solaris and POSIX threads specifications, making those functions significantly slower. Although a program can use `getc_unlocked()` and `putc_unlocked()` for faster access, existing single-threaded programs do not use those functions and would thus be slowed down. Both of these problems are surmountable, but Solaris does not solve them.

- According to the *DCE Porting and Testing Guide*, there are only reference ports of DCE to HP/UX 9.0.1, AIX 3.2.4, OSF/1 1.2, and SINIX 5.41.

- The DCE threads system implements Draft 4 of the POSIX.1c specification. There are significant and subtle differences between Draft 4 and the final approved standard, so code written for DCE threads may encounter subtle incompatibilities with native threads environments on other platforms.

- The DCE threads system apparently does not provide all of the C library routines required by the POSIX standard; for instance, there is no definition of `strtok_r` in the most recently available sources.

- Fundamentally, the DCE threads system appears to be designed to support the DCE environment, and not to provide a full POSIX.1c implementation for general applications.

## 3.4 Debugging tools

For the skilled developer, a user-level threads library such as MIT Pthreads gives the most control over debugging, since the developer has access to all of the internals of the threads system. For the casual developer, however, the lack of integrated debugger support for the threads system can be frustrating. If a thread is preempted during single-stepping, for instance, the debugger will generally fail to notice when the thread resumes. Scheduling threads non-preemptively (an option under MIT Pthreads) can alleviate this problem, but the inability to control which threads are running is still limiting.

The version of `dbx` contained in the Sun development tools for Solaris provides integrated debugging support for threads, including the ability to suspend and resume the execution of individual threads. However, the lack of access to the internals of the threads implementation can make it difficult to debug some problems because it becomes impossible to break at critical points within the implementations of thread-blocking system calls and the like.

# Chapter 4

# Implementation Projects

In evaluating how well multithreaded design applies to the Athena environment, perhaps the most important question is: how does the existing Athena software adapt to a multithreaded design? This question naturally divides itself into two areas:

1. How easy is it to modify the Athena libraries so that they can be used in a multithreaded application?

2. How easy and how useful is it to take an existing Athena application and adapt it to use a multithreaded design?

To answer the first question, I selected two Athena libraries, the Hesiod and Common Error Description libraries, and modified them to be usable in a multithreaded application. To answer the second question, I selected an Athena application, `attach`, and modified it to perform tasks in parallel which it previously did in sequence. For each project I completed, I will discuss the design decisions I made and will judge the success of the project.

## 4.1 The Hesiod Library

Hesiod[1] is a system layered on top of the Domain Name Service (DNS) for distributing database information. In many respects, it fills the same role as Sun Microsystem's Network Information Service[7]. The Hesiod library provides two different kinds of

functionality. First, it provides a function `hes_resolve()` which performs a Hesiod query to a DNS server in much the same fashion as one might perform a hostname lookup, except that the result is a list of text records instead of a host record. Second, the Hesiod library provides functions like `hes_getpwnam()` to emulate C library functions which perform database lookups, using particular Hesiod queries to retrieve the required information. Most of the functions provided by the Hesiod library return data in static buffers.

The first design issue to arise is what to do about the interfaces to the Hesiod library. There are essentially three options:

1. Make the existing interfaces thread-safe by returning results in thread-specific data buffers.

2. Define new interfaces where the user provides data buffers for the returned results. The old interfaces would still exist, but would not be safe to use in multithreaded programs.

3. Define new interfaces with user-provided data buffers as in option 2, and also make the old interfaces thread-safe by returning results in thread-specific data buffers.

Option 1 or 3 require the least number of changes to existing applications which use the Hesiod library, but I chose option 2 for the following reasons:

- Both to reduce code complexity and to make usage easier, it is desirable to be able to compile the library in the same fashion for use in both threaded and non-threaded environments [1]. Therefore, it is preferrable not to use any thread primitives within the Hesiod library.

- Although thread-specific data primitives are provided by POSIX.1c, it may be desirable to use the thread-safe Hesiod library in threads environments with no

[1] Unfortunately, almost all existing environments require all code to be specially compiled for use in a multithreaded program, but in the future there may be environments where this is not true.

thread-specific data facilities. That is, even if we use thread primitives in the library, it is preferrable to avoid the more obscure features of POSIX threads.

- As noted in Chapter 2, returning data in static buffers causes problems even within a single flow of control. It is always better for the user to provide a buffer for results.

- Making an existing application multithreaded usually involves a lot more work than modifying a few Hesiod calls, so trying to save work by making the old interfaces thread-safe isn't a good trade-off.

Thus, I provided new functions `hes_resolve_r()`, `hes_getpwnam_r()`, etc. in the style of POSIX.1c.

A second, similar design decision which came up is how to deal with initialization. The Hesiod library contains several global variables set by reading a configuration file; it is necessary that some thread finish initializing these global variables before any thread performs a Hesiod query. The POSIX.1c standard specifies a function `pthread_once()` to satisfy exactly this kind of constraint, so it would be possible to make sure that the Hesiod library is initialized by having each function begin with a call to `pthread_once()`, but for the same reasons listed in the discussion of the previous design decision, I chose to mandate that the application call `hes_init()` at some point prior to calling `hes_resolve_r()` and other thread-safe functions. Of course, the non-thread-safe interfaces continue to work without a call to `hes_init()`.

A third decision is how to handle errors. Previously, when a Hesiod function failed to finish, the Hesiod library placed an error value indicating the kind of failure in a global variable `Hes_Errno`, which is obviously not appropriate for a thread-safe interface. The POSIX.1c solution for the C library global variable `errno` is to store it in a thread-specific data object, presumably so that they don't have to change the signature of every Unix system call. For the Hesiod library, since I needed to change the signature of every function anyway, and since I wanted to avoid using thread primitives in the library, I chose to have each Hesiod function return its error status as an integer.

A fourth, minor design issue is the interface to `hes_getservbyname_r`, a function which retrieves port numbers by service name. The Hesiod implementation of `getservbyname` involves the possibility of Hesiod query failure conditions (network failures, running out of sockets, etc.) which aren't a possibility in other implementations where this information comes from a local file. Thus, the signature of `hes_getservbyname_r` cannot be the same as the signature of `getservbyname_r`, since it must somehow indicate whether and which Hesiod failure occurred. While this seems like a minor complication, it is an instance of the more general problem of how to deal with implementation-specific errors for an implementation-independent interface.

Finally, there is the issue of the resolver. Hesiod makes use of the `res_send()` resolver function to perform actual queries to the name server. MIT Pthreads provides a thread-safe implementation of `res_send()`, but Solaris does not[2]. Thus, to get the Hesiod library to work in a multithreaded program using native Solaris threads, I was obliged to borrow code from MIT Pthreads to implement the `res_send()` function. Hopefully, better threads environments will eliminate the need for this kind of work-around in the future.

Once these issues had been resolved, only minimal work was required to make the Hesiod library conform to the new specification. Existing functions such as `hes_resolve()` were converted to `hes_resolve_r()` and modified to place return values in result buffers; then, the old functions like `hes_resolve()` were reimplemented in terms of `hes_resolve_r()`. Some effort was required to do length-checking to avoid overwriting the buffers passed in (the old code simply assumed that its static buffers were big enough), but this was not difficult.

To test the new code, I wrote a single-threaded regression test engine (`hestest.c`). It reads lines from a configuration file like:

```
resolve systest group systest:*:17019:
resolve foo bar E
```

[2]Solaris provides a thread-safe resolver interface, but it can only be used for hostname and host address queries.

```
getservbyname zephyr udp zephyr:udp:2102
```

Each line consists of a type of query (`resolve` for a raw query, or the name of a database query), one or two words giving the data to query for, and an expected result (or `E` if an error is expected). The test engine performs lookups using both the thread-safe and non-thread-safe versions of the appropriate functions, compares the actual results to the expected results, and outputs whether each test passed or failed. This design requires that the data file contain the answers to various queries (a couple of answers used in the data file actually changed during the course of writing this thesis), but makes it easy for a developer not familiar with the test suite to determine whether the library passes or fails the tests.

Because there is no interaction within the Hesiod library between threads doing Hesiod lookups, I can have high confidence that the Hesiod library will work properly in a multithreaded program as long as it works properly in a single-threaded program[3]. I also wrote a simple multithreaded hesiod lookup program (`hesthread.c`) as a quick test that the library works for concurrent lookups, but I did not attempt to write a controlled test suite for multithreaded operation.

## 4.2 The Common Error Description Library

The Common Error Description library[5] is a simple tool for building and using tables of errors for different modules of a program. It allows each library or other module of a program to have its own table of errors and error messages; an initialization routine for each module adds its error table to a global list. Once that initialization is done, a single error code can be used to represent either a system error or an error from any module's error table. The application program can then use the function `com_err` to report an error condition given the code for that error.

[3]This is an example of how a multithreaded design can make testing and debugging easier than a design based on an event loop by restricting the logical lifetime of data objects, as I described in Chapter 2. If I had modified the Hesiod library to support an event loop model for concurrent lookups, I would have had to test many different interactions between lookups.

I chose the Common Error Description library for its ubiquitousness rather than its level of challenge; however, a couple of interface issues arose. One involved the function `set_com_err_hook()`, which allows the application to provide an alternative function for handling all `com_err()` calls. Because the whole concept of this function involves global state (the idea being that you can control all error handling in an application without explicitly telling each module how to handle errors), the best way I could find to handle this function is simply to leave things the way they are. A possible alternative involves inherited thread-specific state, but I did not want to introduce thread primitives into the library.

A second, more complicated interface issue arose with the routines for initializing the error table list. The Common Error Description library previously initialized its error table list by automatically generating code for each module which looks like:

```
extern struct et_list *_et_list;
static struct et_list link = { 0, 0 };

void initialize_krb_error_table() {
    if (!link.table) {
        link.next = _et_list;
        link.table = &et;
        _et_list = &link;
    }
}
```

(`et` is a static variable of the source file containing the error table itself.) Pointer assignments are not guaranteed to be atomic by ANSI C or POSIX.1c. Thus, if a `com_err()` call were to occur during an initialization routine, the list might not be consistent at all times. I modified the code to use the ANSI C `sig_atomic_t` type so that the list is always consistent:

```
extern struct et_list *_et_list_ptrs[2];
extern sig_atomic_t _et_list_index;
```

```
static struct et_list link = { 0, 0 };

void initialize_krb_error_table() {
    if (!link.table) {
        link.next = _et_list_ptrs[_et_list_index];
        link.table = &et;
        _et_list_ptrs[!_et_list_index] = &link;
        _et_list_index = !_et_list_index;
    }
}
```

Now the error table list (as defined by `_et_list_ptrs[_et_list_index]`) is always consistent: both of the pointers in `et_list_ptrs` are valid when the list index is updated, and assignments to `sig_atomic_t` are atomic. Thus, it is safe for `com_err` calls to be performed in parallel with an initialization. Of course, it is still not safe for two initialization functions to run in parallel; that problem can only be solved using a lock.

Beyond those decisions, the changes needed to make the Common Error Description library useful to a multithreaded application were trivial. Most of the complexity in the library is in the program for automatically generating the source files for the error tables; this program did not need to be modified except for the change mentioned in the previous paragraph. Two functions needed alternative reentrant interfaces (`error_table_name_r()` and `error_message_r()`), and `com_err` needed to be modified to use those interfaces. Due to the simplicity of the changes made, I performed only cursory tests using the existing test suites for the library, none of which turned up any problems.

## 4.3 The *attach* Program

`attach` is used in the Athena environment to ensure that an Athena "locker" is mounted on the local filesystem of a machine and that the user is authenticated

to the remote filesystem on which the locker resides. `attach` uses a Hesiod query to look up the remote filesystem containing a locker, the local name to use when mounting the locker, and the type of authentication necessary to access the locker. A file `/usr/tmp/attachtab` on each machine keeps track of the lockers attached on that machine. In addition to attaching lockers, `attach` is also capable of detaching lockers, performing maintenance on the `attachtab` file, authenticating to existing lockers, and acquiring Zephyr subscriptions for file server messages about attached lockers.

I picked `attach` because it is representative of Athena software in terms of size (about 8000 lines), complexity (a fairly simple procedurally-oriented design which has been corrupted by a long history of changes), duplication of operating system tasks (in this case, mounting filesystems), and use of Athena libraries. My goal in the project was straightforward: modify the code so that `attach` can attach multiple lockers in parallel.

The first step in making `attach` multithreaded was to introduce function prototypes into the code base. Because most function signatures would have to be changed during the course of the subsequent work, it was vital that the compiler be able to detect calls which hadn't been converted to use new function signatures. (Alternatively, I could have used `lint`, but `lint` is no longer portable.) For convenience, I used the `mkptypes` program, a small utility installed in the Athena system, to generate the required prototypes.

The second step was to analyze the use of global and static data in the program. Use of global data falls into four categories:

1. Configuration parameters. These variables are set during a well-defined phase at the beginning of the program and never changed. In the multithreaded version of the program, these variables can be initialized while the program still has only one thread.

2. Command-line parameters which may differ for each locker to be attached, but don't change during the course of an operation.

3. Variables which potentially change multiple times during the course of attaching a locker, usually error status variables. A few command-line parameters fall into this category when one operation wants to invoke another operation with slightly modified behavior than the usual.

4. Variables containing information shared among the lockers attached, such as the list of Zephyr subscriptions to send after all the lockers are attached.

Variables of the first type could have been left alone, but for internal documentation purposes, I grouped them together in a global structure variable called `config`. Variables of the second and third type must be passed to most functions in the program; I created a `flags` structure containing most of these values and had it passed to most of the functions involved in performing an attach. Variables of the fourth type can be protected by locks.

The most difficult global variable to handle was the `attachtab_first` variable, which points to a list of the entries in the `attachtab` file. For lack of internal documentation, it took considerable time to determine that:

1. Each operation which uses the `attachtab` file separately locks, reads, potentially writes, and unlocks the file, discarding the list acquired from previous reads.

2. No operation keeps the `attachtab` file locked for very long; for example, the operation of attaching a locker first modifies the entry for the locker to indicate that it is attaching the locker, performs the actual attach with the `attachtab` file unlocked, and then modifies the entry to indicate that the lock is attached.

Therefore, it is easiest to treat the list pointed to by `attach_first` as local to a particular attach operation, rather than as data shared between attaches. Once I determined determined this fact, I moved `attach_first` into a structure `attach_list` along with a few other variables used to keep track of the in-memory list of `attachtab` entries. I did not use the `flags` structure for these variables, since there are a few cases where multiple copies of the `attachtab` entries are kept in memory at one time,

such as during the operation of detaching all unwanted lockers in the `attachtab` file[4].

A third step in the conversion process was output handling. Under some circumstances, `attach` is required to print its output in an order corresponding to the order of the locker names passed on the command line. I provided an output buffer in the `flags` structure and had the high-level `attachcmd()` function output the contents of the output buffer after each attach operation completed, in order of the command-line arguments. I left output to standard error alone, so error output might be reordered, on the assumption that the ordering of error output is not crucial to any uses of `attach`.

Converting `attach` into a multithreaded program was very time-consuming. In addition to the above issues relating to use of global data, many minor issues arose: I had to disable support for the RVD filesystem because of the antiquated system interface; I had to lock around calls to the RPC subsystem, the Zephyr library, and the Kerberos library (leading to the inability to perform multiple such operations in parallel); and I was unable to efficiently perform the operation where `attach` uses `setuid` to test the user's ability to read an AFS path, since user credentials are global to a process. The use of many different system services made `attach` a difficult program to multithread.

I was able to get `attach` to work properly under both MIT Pthreads and Solaris threads. I measured only slight performance benefits from doing attach operations in parallel rather than serially, and contention for the `attachtab` file often increased the overall running time. I saw several opportunities for speeding up the process of attaching multiple lockers, but none specific to a multithreaded program.

[4]The acute reader may wonder how the original `attach` program used the same global variable to contain two copies of the list of `attachtab` entries. The answer is that, in such cases, the `attach` program would copy the value of `attachtab_first` to a local variable and resets `attachtab_first` to `NULL` before getting the second copy of the list. I rejected this approach as too inelegant.

# Chapter 5

# Conclusions

As a design tool, the use of multiple threads should be treated with caution in the Athena environment. Development environments for threads are still primitive and nonstandardized, with native POSIX-compliant threads environments only recently emerging. Programs in the Athena environment tend to use a wide range of system services; if any of these services cannot be used in a multithreaded program, the robustness of a multithreaded design for that program could be seriously compromised.

On the other hand, as I illustrated in two of my implementation projects, it is not difficult to convert Athena libraries to be usable in a multithreaded program without wedding ourselves to multithreaded development environments. Furthermore, as I noted in Chapter 2, interfaces which return data in static buffers create problems in single-threaded as well as multi-threaded programs. It would be in our interest to convert more of the Athena libraries to support thread-safe interfaces, both because of the cleaner design such interfaces entail and because it would help make multithreaded design an available tool when multithreaded development environments become more useful.

# Appendix A

# Code

This appendix contains the modified code that I produced in the course of this thesis. Obviously, the code that is here is (with the exception of the Hesiod test driver and `hesthread.c`) based on code written by other people. Please refer to Chapter 4 to determine what parts of the design are my contribution.

## A.1 Hesiod

### A.1.1 Imakefile

```
# Makefile for the Project Athena Hesiod Nameserver library
#
# $Id: Imakefile,v 1.1 1995/03/29 00:41:09 ghudson Exp ghudson $
#

#if defined(_AIX) && defined(_IBMR2)
OSDEF= -DBIT_ZERO_ON_LEFT
#endif

#ifdef SOLARIS
OSDEF= -D_REENTRANT -DINTERNAL_RESOLVER -Isolaris
OSSRC= hes_internal.c hes_send.c hes_mkquery.c
OSOBJ= hes_internal.o hes_send.o hes_mkquery.o
LDLIBS= -lsocket -lnsl -lthread
#else
CC=pgcc
LD=pgcc
```

```
#endif
CDEBUG=-g

DEFINES=-I. -DHESIOD $(OSDEF)

SRCS = hesiod.c hespwnam.c hesmailhost.c hesservbyname.c resolve.c cistrcmp.c \
$(OSOBJ)
OBJS = hesiod.o hespwnam.o hesmailhost.o hesservbyname.o resolve.o cistrcmp.o \
$(OSOBJ)

ProfiledObjectRule()

SimpleLibrary(hesiod,$(OBJS),$(ATHLIBDIR))

SimpleProgram(hesinfo,hesinfo.o libhesiod.a,,$(ATHRBINDIR))
build_program(hestest,hestest.o libhesiod.a,,)

install_man(hesiod.3,hesiod.3)
install_man(hesinfo.1,hesinfo.1)

install_file(hesiod.h,$(ATHINCDIR)/hesiod.h)

$(OBJS):: hesiod.h

hesiod.h: hesiod.h.sed
$(RM) hesiod.h
sed s:CONFDIR:$(ATHCONFDIR): < hesiod.h.sed > hesiod.h

check: all
./hestest hestest.conf
```

### A.1.2 hesiod.h.sed

```
/* This file contains definitions for use by the Hesiod name service and
 * applications.
 *
 * For copying and distribution information, see the file <mit-copyright.h>.
 *
 * Original version by Steve Dyer, IBM/Project Athena.
 *
 *      $Id: hesiod.h.sed,v 1.1 1995/03/18 16:38:25 ghudson Exp ghudson $
 */
```

10

```
#ifndef _HESIOD_H
#define _HESIOD_H
```

```
/* Configuration information. */

#define HESIOD_CONF "CONFDIR/hesiod.conf"  /* Configuration file. */
#define DEF_RHS     ".Athena.MIT.EDU"      /* Defaults if HESIOD_CONF */
#define DEF_LHS     ".ns"                  /*    file is not present. */

/* Error codes. */                                                    20

#define HES_ER_UNINIT -1    /* uninitialized */
#define HES_ER_OK     0     /* no error */
#define HES_ER_NOTFOUND 1   /* Hesiod name not found by server */
#define HES_ER_CONFIG 2     /* local problem (no config file?) */
#define HES_ER_NET    3     /* network problem */
#define HES_ER_RANGE 4      /* return buffer not large enough */
#define HES_ER_NOMEM 5      /* insufficient memory is available */
#define HES_ER_INVAL 6      /* invalid response from hesiod server */
                                                                      30
/* For use in getting post-office information. */

struct hes_postoffice {
        char    *po_type;
        char    *po_host;
        char    *po_name;
};

/* Declaration of routines */
                                                                      40
#ifdef _STDC_

/* Thread-safe. */
int hes_init(void);
int hes_to_bind_r(char *HesiodName, char *HesiodNameType,
                  char *buffer, int buflen);
int hes_resolve_r(char *HesiodName, char *HesiodNameType,
                  char **retvec, int retveclen);
int hes_getpwnam_r(char *nam, struct passwd *entry, char *buf, int bufsize);
int hes_getpwuid_r(int uid, struct passwd *entry, char *buf, int bufsize);   50
int hes_getmailhost_r(char *user, struct hes_postoffice *ret,
                      char *linebuf, int bufsize);
int hes_getservbyname_r(char *name, char *proto, struct servent *result,
                        char *buffer, int buflen);

/* Non-thread-safe. */
char *hes_to_bind(char *HesiodName, char *HesiodNameType);
char **hes_resolve(char *HesiodName, char *HesiodNameType);
int hes_error(void);
struct passwd *hes_getpwnam(char *nam);                                   60
struct passwd *hes_getpwuid(int uid);
struct hes_postoffice *hes_getmailhost(char *user);
struct servent *hes_getservbyname(char *name, char *proto);

#else

char *hes_to_bind(), **hes_resolve();
```

```
int hes_init(), hes_error(), hes_to_bind_r(), hes_resolve_r();
int hes_getmailhost_r(), hes_getservbyname_r();
int hes_getpwnam_r(), hes_getpwuid_r();
struct hes_postoffice *hes_getmailhost();
struct servent *hes_getservbyname();
struct passwd *hes_getpwnam(), *hes_getpwuid();

#endif /* _STDC_ */

#endif /* _HESIOD_H */
```

## A.1.3 hes_internal.h

```
/*
 * Copyright (c) 1985 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *        notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *        notice, this list of conditions and the following disclaimer in the
 *        documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *        must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *        may be used to endorse or promote products derived from this software
 *        without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.    IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#if defined(LIBC_SCCS) && !defined(lint)
/*static char *sccsid = "from: @(#)hes_internal.h          6.22 (Berkeley) 3/19/91";*/
static char *rcsid = "$Id:  hes_internal.h,v 1.7 1995/03/18 05:52:03 ghudson Exp $";
#endif /* LIBC_SCCS and not lint */

#ifndef _HES_INTERNAL_H
```

```
#define _HES_INTERNAL_H                                                    40

#include <arpa/nameser.h>
#include <netinet/in.h>
#include <netdb.h>
#include <resolv.h>

#define MAXRESOLVSORT    10      /* number of net to sort on */
#define MAXDNSLUS        4       /* max # of host lookup types */
#define HES_TIMEOUT      5       /* min. seconds between retries */
#define HES_MAXNDOTS     15      /* should reflect bit field size */       50

/*
 * Resolver options
 */
#define HES_INIT         0x0001          /* address initialized */
#define HES_DEBUG        0x0002          /* print debug messages */
#define HES_AAONLY       0x0004          /* authoritative answers only */
#define HES_USEVC        0x0008          /* use virtual circuit */
#define HES_PRIMARY      0x0010          /* query primary server only */
#define HES_IGNTC        0x0020          /* ignore trucation errors */     60
#define HES_RECURSE      0x0040          /* recursion desired */
#define HES_DEFNAMES     0x0080          /* use default domain name */
#define HES_STAYOPEN     0x0100          /* Keep TCP socket open */
#define HES_DNSRCH       0x0200          /* search up local domain tree */

#define HES_DEFAULT  (HES_RECURSE | HES_DEFNAMES | HES_DNSRCH)

struct _hes_state {
        int     retrans;                /* retransmition time interval */
        int     retry;                  /* number of times to retransmit */   70
        long    options;                /* option flags - see below. */
        int     nscount;                /* number of name servers */
        struct  sockaddr_in nsaddr_list[MAXNS];    /* address of name server */
#define nsaddr  nsaddr_list[0]          /* for backward compatibility */
        u_short id;                     /* current packet id */
        char    *dnsrch[MAXDNSRCH+1]; /* components of domain to search */
        char    defdname[MAXDNAME];   /* default domain */
        long    pfcode;                 /* HES_PRF_ flags - see below. */
        u_char  ndots:4;                /* threshold for initial abs. query */
        u_char  nsort:4;                /* number of elements in sort_list[] */   80
        char    unused[3];
        struct {
                struct in_addr addr;
                u_long mask;
        } sort_list[MAXRESOLVSORT];
        char    lookups[MAXDNSLUS];
};

struct  hes_data {
        struct _hes_state state;                                          90
        int errval;
        int sock;
        FILE *hostfp;
```

```
        int keep_hostfp_open;
};

struct hes_data *_hes_init(void);

#endif
```

## A.1.4 mit-copyright.h

```
/* $Header: mit-copyright.h,v 1.3 88/08/07 21:52:57 treese Exp $ */
/*

Copyright 1988 by the Massachusetts Institute of Technology.

Permission to use, copy, modify, and distribute this
software and its documentation for any purpose and without
fee is hereby granted, provided that the above copyright
notice appear in all copies and that both that copyright
notice and this permission notice appear in supporting
documentation, and that the name of M.I.T. not be used in
advertising or publicity pertaining to distribution of the
software without specific, written prior permission.
M.I.T. makes no representations about the suitability of
this software for any purpose.  It is provided "as is"
without express or implied warranty.

*/
```

## A.1.5 resscan.h

```
/* This file contains definitions for the name resolver scanning routines
 * used by the Hesiod library.
 *
 * For copying and distribution information, see the file <mit-copyright.h>.
 *
 * Original version by Steve Dyer, IBM/Project Athena, and Sam Hsu,
 * DEC/Project Athena.
 *
 *      $Author: ghudson $
 *      $Athena: resscan.h,v 1.3 88/08/07 21:53:09 treese Exp $
 *      $Header: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/resscan.h,v 1.1 1995/03/18 17:
 *      $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/resscan.h,v $
 *      $Log: resscan.h,v $
 * Revision 1.1  1995/03/18  17:39:00  ghudson
 * Initial revision
 *
 * Revision 1.3  88/08/07  21:53:09  treese
 * First public distribution
 *
 * Revision 1.2  88/06/12  00:53:14  treese
 * Cleaned up to work with Saber.
```

```
 * First public distribution.
 *
 * Revision 1.2  88/06/05  19:51:32  treese
 * Cleaned up for public distribution
 *
 */

typedef struct rr {
    short type;                 /* RR type */
    short class;                /* RR class */
    int dlen;                   /* len of data section */
    char *data;                 /* pointer to data */
} rr_t, *rr_p;

typedef struct nsmsg {
    int len;                    /* sizeof(msg) */
    int ns_off;                 /* offset to name server RRs */
    int ar_off;                 /* offset to additional RRs */
    int count;                  /* total number of RRs */
    HEADER *hd;                 /* message header */
    rr_t rr;                    /* vector of (stripped-down) RR descriptors */
} nsmsg_t, *nsmsg_p;

typedef struct retransXretry {
    short retrans;
    short retry;
} retransXretry_t, *retransXretry_p;


#define NMSGSIZE (sizeof(nsmsg_t) + \
                  sizeof(rr_t) * ((PACKETSZ - sizeof(HEADER)) / RRFIXEDSZ))
#define DATASIZE (PACKETSZ - sizeof(HEADER))

extern char *p_cdname(), *p_rr(), *p_type(), *p_class();
extern struct nsmsg *res_scan(), *resolve(), *_resolve();
```

## A.1.6 solaris/pthread.h

```
/* System includes for Solaris threads. */
#include <thread.h>
#include <synch.h>

/* Constants and types. */
#define _POSIX_THREADS
#define _POSIX_THREAD_ATTR_STACKSIZE
#define _POSIX_THREAD_ATTR_STACKADDR
#define PTHREAD_STACK_MIN               thr_min_stack()
#define PTHREAD_CREATE_DETACHED         THR_DETACHED
#define PTHREAD_CREATE_JOINABLE         0
#define PTHREAD_MUTEX_INITIALIZER       SHAREDMUTEX
#define PTHREAD_COND_INITIALIZER        SHAREDCV
#define PTHREAD_ONCE_INIT               { PTHREAD_MUTEX_INITIALIZER, 0 }
typedef thread_t pthread_t;
```

```
typedef mutex_t pthread_mutex_t;
typedef cond_t pthread_cond_t;
typedef thread_key_t pthread_key_t;
typedef struct { int size; void *base; int flags; } pthread_attr_t;
typedef struct { pthread_mutex_t lock; int flag; } pthread_once_t;

/* Translations for attribute initialization and thread creation. */
#define pthread_attr_init(a)        ((a)->base = 0, (a)->size = (a)->flags = 0)
#define pthread_attr_destroy(a)
#define pthread_attr_setstacksize(a, b)        ((a)->size = (b))
#define pthread_attr_getstacksize(a)      ((a)->size)
#define pthread_attr_setstackaddr(a, b)   ((a)->base = (b))
#define pthread_attr_getstackaddr(a)      ((a)->base)
#define pthread_attr_setdetachstate(a, b) ((a)->flags = (b))
#define pthread_attr_getdetachstate(a)    ((a)->flags)
#define pthread_create(a, b, c, d) \
        thr_create((b)->base, (b)->size, c, d, (b)->flags, a)

/* Straightforward translations for mutexes and conditions. */
#define pthread_mutex_init(a, b)        mutex_init(a, USYNC_THREAD, 0)
#define pthread_mutex_destroy             mutex_destroy
#define pthread_mutex_lock                mutex_lock
#define pthread_mutex_unlock              mutex_unlock
#define pthread_mutex_trylock             mutex_trylock
#define pthread_cond_init(a, b)           cond_init(a, USYNC_THREAD, 0)
#define pthread_cond_destroy            cond_destroy
#define pthread_cond_wait               cond_wait
#define pthread_cond_signal             cond_signal
#define pthread_cond_broadcast           cond_broadcast

/* Doing things once. */
#define pthread_once(once, func) { \
        if (!(once)->flag) { \
                pthread_mutex_lock(&(once)->lock); \
                if (!(once)->flag) { \
                        (*(func))(); \
                        (once)->flag = 1; \
                } \
                pthread_mutex_unlock(&(once)->lock); \
        } \
}

/* Thread-specific data. */
#define pthread_key_create              thr_keycreate
#define pthread_setspecific             thr_setspecific
static void *pthread_getspecific(pthread_key_t key) {
   void *value;

   return (thr_getspecific(key, &value) == 0) ? value : NULL;
}
```

## A.1.7 cistrcmp.c

```
#ifndef lint
static char sccsid[] = "@(#)cistrcmp.c    4.3 (Berkeley) 6/4/86";
#endif

/*
 * Copyright (c) 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

/*
 * This array is designed for mapping upper and lower case letter
 * together for a case independent comparison.  The mappings are
 * based upon ascii character sequences.
 */

static char charmap[] = {
        '\000', '\001', '\002', '\003', '\004', '\005', '\006', '\007',
        '\010', '\011', '\012', '\013', '\014', '\015', '\016', '\017',
        '\020', '\021', '\022', '\023', '\024', '\025', '\026', '\027',
        '\030', '\031', '\032', '\033', '\034', '\035', '\036', '\037',
        '\040', '\041', '\042', '\043', '\044', '\045', '\046', '\047',
        '\050', '\051', '\052', '\053', '\054', '\055', '\056', '\057',
        '\060', '\061', '\062', '\063', '\064', '\065', '\066', '\067',
        '\070', '\071', '\072', '\073', '\074', '\075', '\076', '\077',
        '\100', '\141', '\142', '\143', '\144', '\145', '\146', '\147',
        '\150', '\151', '\152', '\153', '\154', '\155', '\156', '\157',
        '\160', '\161', '\162', '\163', '\164', '\165', '\166', '\167',
        '\170', '\171', '\172', '\133', '\134', '\135', '\136', '\137',
        '\140', '\141', '\142', '\143', '\144', '\145', '\146', '\147',
        '\150', '\151', '\152', '\153', '\154', '\155', '\156', '\157',
        '\160', '\161', '\162', '\163', '\164', '\165', '\166', '\167',
        '\170', '\171', '\172', '\173', '\174', '\175', '\176', '\177',
        '\200', '\201', '\202', '\203', '\204', '\205', '\206', '\207',
        '\210', '\211', '\212', '\213', '\214', '\215', '\216', '\217',
        '\220', '\221', '\222', '\223', '\224', '\225', '\226', '\227',
        '\230', '\231', '\232', '\233', '\234', '\235', '\236', '\237',
        '\240', '\241', '\242', '\243', '\244', '\245', '\246', '\247',
        '\250', '\251', '\252', '\253', '\254', '\255', '\256', '\257',
        '\260', '\261', '\262', '\263', '\264', '\265', '\266', '\267',
        '\270', '\271', '\272', '\273', '\274', '\275', '\276', '\277',
        '\300', '\341', '\342', '\343', '\344', '\345', '\346', '\347',
        '\350', '\351', '\352', '\353', '\354', '\355', '\356', '\357',
        '\360', '\361', '\362', '\363', '\364', '\365', '\366', '\367',
        '\370', '\371', '\372', '\333', '\334', '\335', '\336', '\337',
        '\340', '\341', '\342', '\343', '\344', '\345', '\346', '\347',
        '\350', '\351', '\352', '\353', '\354', '\355', '\356', '\357',
        '\360', '\361', '\362', '\363', '\364', '\365', '\366', '\367',
        '\370', '\371', '\372', '\373', '\374', '\375', '\376', '\377',
};

cistrcmp(s1, s2)
register char *s1, *s2;
{
```

```
        register char *cm = charmap;

        while (cm[*s1] == cm[*s2++])
                if (*s1++=='\0')
                        return(0);
        return(cm[*s1] - cm[*--s2]);
}

cistrncmp(s1, s2, n)
register char *s1, *s2;
register n;
{
        register char *cm = charmap;

        while (--n >= 0 && cm[*s1] == cm[*s2++])
                if (*s1++ == '\0')
                        return(0);
        return(n<0 ? 0 : cm[*s1] - cm[*--s2]);
}
```

## A.1.8 hes_internal.c

```
/*
 * Copyright (c) 1985 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *        notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *        notice, this list of conditions and the following disclaimer in the
 *        documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *        must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *        may be used to endorse or promote products derived from this software
 *        without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSI
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
```

```
 */

#if defined(LIBC_SCCS) && !defined(lint)
/*static char *sccsid = "from: @(#)hes_internal.c          6.22 (Berkeley) 3/19/91";*/
static char *rcsid = "$Id:  hes_internal.c,v 1.13 1995/09/20 21:04:28 ghudson Exp $";
#endif /* LIBC_SCCS and not lint */

#include <pthread.h>
#include <sys/types.h>                                                    40
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <string.h>
#include "hes_internal.h"

#define DEFAULT_RETRIES 4

static pthread_once_t init_once = PTHREAD_ONCE_INIT;                      50
static int init_status;
static pthread_key_t key;

static void _hes_init_global();
static void set_options(const char *options, const char *source);
static unsigned int net_mask(struct in_addr in);
static int qcomp(const void *arg1, const void *arg2);

static struct _hes_state start;
                                                                          60
/* Performs global initialization. */
struct hes_data *_hes_init()
{
        struct hes_data *data;

        /* Make sure the global initializations have been done. */
        pthread_once(&init_once, _hes_init_global);
        if (init_status < 0)
                return NULL;
                                                                          70
        /* Initialize thread-specific data for this thread if it hasn't
         * been done already. */
        data = (struct hes_data *) pthread_getspecific(key);
        if (!data) {
                data = (struct hes_data *) malloc(sizeof(struct hes_data));
                if (data == NULL)
                        return NULL;
                if (pthread_setspecific(key, data) < 0) {
                        free(data);
                        return NULL;                                      80
                }
                data->state = start;
                data->errval = NO_RECOVERY;
                data->sock = -1;
                data->hostfp = NULL;
```

```
                data->keep_hostfp_open = 0;
        }
        return data;
}

static void _hes_init_global()
{
        int result;
        char line[BUFSIZ], buf[BUFSIZ], *domain, *p, *net;
        int i, localdomain_set = 0, num_servers = 0, num_sorts = 0;
        FILE *fp;
        struct in_addr addr;

        /* Assume an error state until we finish. */
        init_status = -1;

        /* Initialize the key for thread-specific data. */
        result = pthread_key_create(&key, free);
        if (result < 0)
                return;

        /* Initialize starting state. */
        start.retrans = HES_TIMEOUT;
        start.retry = DEFAULT_RETRIES;
        start.options = HES_DEFAULT;
        start.id = 0;
        start.nscount = 1;
        start.nsaddr.sin_addr.s_addr = INADDR_ANY;
        start.nsaddr.sin_family = AF_INET;
        start.nsaddr.sin_port = htons(NAMESERVER_PORT);
        start.nscount = 1;
        start.ndots = 1;
        start.pfcode = 0;
        strncpy(start.lookups, "f", sizeof(start.lookups));

        /* Look for a LOCALDOMAIN definition. */
        domain = getenv("LOCALDOMAIN");
        if (domain != NULL) {
                strncpy(start.defdname, domain, sizeof(start.defdname));
                domain = start.defdname;
                localdomain_set = 1;

                /* Construct a search path from the LOCALDOMAIN value, which is
                 * a space-separated list of strings.  For backwards-compatibility,
                 * a newline terminates the list. */
                i = 0;
                while (*domain && i < MAXDNSRCH) {
                        start.dnsrch[i] = domain;
                        while (*domain && !isspace(*domain))
                                domain++;
                        if (!*domain || *domain == '\n') {
                                *domain = 0;
                                break;
                        }
```

```
                *domain++ = 0;
                while (isspace(*domain))
                        domain++;
                i++;
        }
}

/* Look for a config file and read it in. */
fp = fopen(_PATH_RESCONF, "r");
if (fp != NULL) {
        strncpy(start.lookups, "bf", sizeof(start.lookups));

        /* Read in the configuration file. */
        while (fgets(line, sizeof(line), fp)) {

                /* Ignore blank lines and comments. */
                if (*line == ';' || *line == '#' || !*line)
                        continue;

                if (strncmp(line, "domain", 6) == 0) {

                        /* Read in the default domain, and initialize a one-
                         * element search path.      Skip the domain line if we
                         * already got one from the LOCALDOMAIN environment
                         * variable. */
                        if (localdomain_set)
                                continue;

                        /* Look for the next word in the line. */
                        p = line + 6;
                        while (*p == ' ' || *p == '\t')
                                p++;
                        if (!*p || *p == '\n')
                                continue;

                        /* Copy in the domain, and null-terminate it at the
                         * first tab or newline. */
                        strncpy(start.defdname, p, sizeof(start.defdname) - 1);
                        p = strpbrk(start.defdname, "\t\n");
                        if (p)
                                *p = 0;

                        start.dnsrch[0] = start.defdname;
                        start.dnsrch[1] = NULL;

                } else if (strncmp(line, "lookup", 6) == 0) {

                        /* Get a list of lookup types. */
                        memset(start.lookups, 0, sizeof(start.lookups));

                        /* Find the next word in the line. */
                        p = line + 6;
                        while (isspace(*p))
                                p++;
```

```
        i = 0;
        while (*p && i < MAXDNSLUS) {
                /* Add a lookup type. */
                if (*p == 'y' || *p == 'b' || *p == 'f')
                        start.lookups[i++] = *p;

                /* Find the next word. */
                while (*p && !isspace(*p))
                        p++;
                while (isspace(*p))
                        p++;
        }

} else if (strncmp(line, "search", 6) == 0) {

        /* Read in a space-separated list of domains to search
         * when a name is not fully-qualified.  Skip this line
         * if the LOCALDOMAIN environment variable was set. */
        if (localdomain_set)
                continue;

        /* Look for the next word on the line. */
        p = line + 6;
        while (*p == ' ' || *p == '\t')
                p++;
        if (!*p || *p == '\n')
                continue;

        /* Copy the rest of the line into start.defdname. */
        strncpy(start.defdname, p, sizeof(start.defdname) - 1);
        domain = start.defdname;
        p = strchr(domain, '\n');
        if (*p)
                *p = 0;

        /* Construct a search path from the line, which is a
         * space-separated list of strings. */
        i = 0;
        while (*domain && i < MAXDNSRCH) {
                start.dnsrch[i] = domain;
                while (*domain && !isspace(*domain))
                        domain++;
                if (!*domain || *domain == '\n') {
                        *domain = 0;
                        break;
                }
                *domain++ = 0;
                while (isspace(*domain))
                        domain++;
                i++;
        }

} else if (strncmp(line, "nameserver", 10) == 0) {
```

```
                /* Add an address to the list of name servers we can
                 * connect to. */
                /* Look for the next word in the line. */
                p = line + 10;
                while (*p == ' ' || *p == '\t')
                        p++;
                if (*p && *p != '\n') {
                        addr.s_addr = inet_addr(p);
                        start.nsaddr_list[num_servers].sin_addr = addr;
                        start.nsaddr_list[num_servers].sin_family = AF_INET;
                        start.nsaddr_list[num_servers].sin_port =
                                htons(NAMESERVER_PORT);
                        if (++num_servers >= MAXNS)
                           break;
                }

        } else if (strncmp(line, "sortlist", 8) == 0) {

                p = line + 8;
                while (num_sorts < MAXRESOLVSORT) {
                        /* Find the next word in the line. */
                        p = line + 8;
                        while (*p == ' ' || *p == '\t')
                                p++;

                        /* Read in an IP address and netmask. */
                        if (sscanf(p, "%[0-9./]s", buf) != 1)
                                break;
                        net = strchr(buf, '/');
                        if (net)
                                *net = 0;

                        /* Translate the address into an IP address
                         * and netmask. */
                        addr.s_addr = inet_addr(buf);
                        start.sort_list[num_sorts].addr = addr;
                        if (net) {
                                addr.s_addr = inet_addr(net + 1);
                                start.sort_list[num_sorts].mask = addr.s_addr;
                        } else {
                                start.sort_list[num_sorts].mask =
                                        net_mask(start.sort_list[num_sorts].addr);
                        }
                        num_sorts++;

                        /* Skip past this word. */
                        if (net)
                                *net = '/';
                        p += strlen(buf);
                }
```

```
                }
        }
        fclose(fp);
}

/* If we don't have a default domain, strip off the first
 * component of this machine's domain name, and make a one-
 * element search path consisting of the default domain. */
if (*start.defdname == 0) {                                          310
        if (gethostname(buf, sizeof(start.defdname) - 1) == 0) {
                p = strchr(buf, '.');
                if (p)
                        strcpy(start.defdname, p + 1);
        }
        start.dnsrch[0] = start.defdname;
        start.dnsrch[1] = NULL;
}

p = getenv("RES_OPTIONS");                                           320
if (p)
        set_options(p, "env");

start.options |= HES_INIT;
init_status = 0;
}

static void set_options(const char *options, const char *source)
{
        const char *p = options;                                     330
        int i;

        while (*p) {

                /* Skip leading and inner runs of spaces. */
                while (*p == ' ' || *p == '\t')
                        p++;

                /* Search for and process individual options. */
                if (strncmp(p, "ndots:", 6) == 0) {                  340
                        i = atoi(p + 6);
                        if (i <= HES_MAXNDOTS)
                                start.ndots = i;
                        else
                                start.ndots = HES_MAXNDOTS;
                } else if (!strncmp(p, "debug", 5))
                   start.options |= HES_DEBUG;
                else if (!strncmp(p, "usevc", 5))
                   start.options |= HES_USEVC;
                else if (!strncmp(p, "stayopen", 8))                350
                   start.options |= HES_STAYOPEN;

                /* Skip to next run of spaces */
                while (*p && *p != ' ' && *p != '\t')
                        p++;
```

```
        }
}

static unsigned int net_mask(struct in_addr in)
{                                                                    360
        unsigned int i = ntohl(in.s_addr);

        if (IN_CLASSA(i))
                return htonl(IN_CLASSA_NET);
        if (IN_CLASSB(i))
                return htonl(IN_CLASSB_NET);
        return htonl(IN_CLASSC_NET);
}

static int qcomp(const void *arg1, const void *arg2)                370
{
        const struct in_addr **a1 = (const struct in_addr **) arg1;
        const struct in_addr **a2 = (const struct in_addr **) arg2;
        struct hes_data *data = (struct hes_data *) pthread_getspecific(key);
        struct _hes_state *state = &data->state;
        int pos1, pos2;

        for (pos1 = 0; pos1 < state->nsort; pos1++) {
                if (state->sort_list[pos1].addr.s_addr ==
                        ((*a1)->s_addr & state->sort_list[pos1].mask))   380
                        break;
        }
        for (pos2 = 0; pos2 < state->nsort; pos2++) {
                if (state->sort_list[pos2].addr.s_addr ==
                        ((*a2)->s_addr & state->sort_list[pos2].mask))
                        break;
        }
        return pos1 - pos2;
}
                                                                     390
/*
 * Compress domain name 'exp_dn' into 'comp_dn'.
 * Return the size of the compressed name or -1.
 * 'length' is the size of the array pointed to by 'comp_dn'.
 * 'dnptrs' is a list of pointers to previous compressed names. dnptrs[0]
 * is a pointer to the beginning of the message. The list ends with NULL.
 * 'lastdnptr' is a pointer to the end of the arrary pointed to
 * by 'dnptrs'. Side effect is to update the list of pointers for
 * labels inserted into the message as we compress the name.
 * If 'dnptr' is NULL, we don't try to compress names. If 'lastdnptr'  400
 * is NULL, we don't update the list.
 */
int
dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
        const u_char *exp_dn;
        u_char *comp_dn, **dnptrs, **lastdnptr;
        int length;
{
        register u_char *cp, *dn;
```

```
register int c, l;                                                    410
u_char **cpp, **lpp, *sp, *eob;
u_char *msg;

dn = (u_char *)exp_dn;
cp = comp_dn;
eob = cp + length;
if (dnptrs != NULL) {
        if ((msg = *dnptrs++) != NULL) {
                for (cpp = dnptrs; *cpp != NULL; cpp++)
                        ;                                             420
                lpp = cpp;                          /* end of list to search */
        }
} else
        msg = NULL;
for (c = *dn++; c != '\0'; ) {
        /* look to see if we can use pointers */
        if (msg != NULL) {
                if ((l = dn_find(dn-1, msg, dnptrs, lpp)) >= 0) {
                        if (cp+1 >= eob)
                                return (-1);                          430
                        *cp++ = (l >> 8) | INDIR_MASK;
                        *cp++ = l % 256;
                        return (cp - comp_dn);
                }
                /* not found, save it */
                if (lastdnptr != NULL && cpp < lastdnptr-1) {
                        *cpp++ = cp;
                        *cpp = NULL;
                }
        }                                                             440
        sp = cp++;                                  /* save ptr to length byte */
        do {
                if (c == '.') {
                        c = *dn++;
                        break;
                }
                if (c == '\\') {
                        if ((c = *dn++) == '\0')
                                break;
                }                                                     450
                if (cp >= eob) {
                        if (msg != NULL)
                                *lpp = NULL;
                        return (-1);
                }
                *cp++ = c;
        } while ((c = *dn++) != '\0');
        /* catch trailing '.'s but not '..' */
        if ((l = cp - sp - 1) == 0 && c == '\0') {
                cp--;                                                 460
                break;
        }
        if (l <= 0 || l > MAXLABEL) {
```

```
                        if (msg != NULL)
                                *lpp = NULL;
                        return (-1);
                }
                *sp = l;
        }
        if (cp >= eob) {                                                    470
                if (msg != NULL)
                        *lpp = NULL;
                return (-1);
        }
        *cp++ = '\0';
        return (cp - comp_dn);
}

/*
 * Search for expanded name from a list of previously compressed names.      480
 * Return the offset from msg if found or -1.
 * dnptrs is the pointer to the first name on the list,
 * not the pointer to the start of the message.
 */
static int
dn_find(exp_dn, msg, dnptrs, lastdnptr)
        u_char *exp_dn, *msg;
        u_char **dnptrs, **lastdnptr;
{
        register u_char *dn, *cp, **cpp;                                    490
        register int n;
        u_char *sp;

        for (cpp = dnptrs; cpp < lastdnptr; cpp++) {
                dn = exp_dn;
                sp = cp = *cpp;
                while (n = *cp++) {
                        /*
                         * check for indirection
                         */                                                 500
                        switch (n & INDIR_MASK) {
                          case 0:                       /* normal case, n == len */
                                while (--n >= 0) {
                                        if (*dn == '.')
                                                goto next;
                                        if (*dn == '\\')
                                                dn++;
                                        if (*dn++ != *cp++)
                                                goto next;
                                }                                           510
                                if ((n = *dn++) == '\0' && *cp == '\0')
                                        return (sp - msg);
                                if (n == '.')
                                        continue;
                                goto next;

                          default:                      /* illegal type */
```

```
                        return (-1);

                    case INDIR_MASK:        /* indirection */
                        cp = msg + (((n & 0x3f) << 8) | *cp);
                    }
                }
                if (*dn == '\0')
                    return (sp - msg);
            next:   ;
        }
        return (-1);
}

/*
 * Routines to insert/extract short/long's. Must account for byte
 * order and non-alignment problems. This code at least has the
 * advantage of being portable.
 *
 * used by sendmail.
 */

u_short _getshort(register const u_char *msgp)
{
        register u_short u;

        GETSHORT(u, msgp);
        return (u);
}

u_long _getlong(const u_char *msgp)
{
        int u;

        GETLONG(u, msgp);
        return (u);
}

void _putshort(register u_short s, register u_char *msgp)
{
        PUTSHORT(s, msgp);
}

void _putlong(u_long l, u_char *msgp)
{
        PUTLONG(l, msgp);
}
```

## A.1.9 hes_mkquery.c

```
/*
 * Copyright (c) 1985 Regents of the University of California.
 * All rights reserved.
```

```
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *        notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *        notice, this list of conditions and the following disclaimer in the
 *        documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *        must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *        may be used to endorse or promote products derived from this software
 *        without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#if defined(LIBC_SCCS) && !defined(lint)
/*static char *sccsid = "from: @(#)hes_mkquery.c          6.16 (Berkeley) 3/6/91";*/
static char *rcsid = "$Id:  hes_mkquery.c,v 1.6 1995/02/12 04:50:46 snl Exp $";
#endif /* LIBC_SCCS and not lint */

#include <sys/param.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include "hes_internal.h"

/*
 * Form all types of queries.
 * Returns the size of the result or -1.
 */
hes_mkquery(op, dname, class, type, data, datalen, newrr_in, buf, buflen)
        int op;                 /* opcode of query */
        const char *dname;              /* domain name */
        int class, type;        /* class and type of query */
        const char *data;               /* resource record data */
        int datalen;            /* length of data */
```

```
        const char *newrr_in;       /* new rr for modify or append */
        char *buf;                  /* buffer to put query */
        int buflen;                 /* size of buffer */                    60
{
        register HEADER *hp;
        register char *cp;
        register int n;
        struct rrec *newrr = (struct rrec *) newrr_in;
        char *dnptrs[10], **dpp, **lastdnptr;
        struct hes_data *hdata;
        struct _hes_state *_rs;

        /*                                                                  70
         * Initialize header fields.
         */

        hdata = _hes_init();
        if (!hdata)
                return -1;
        _rs = &hdata->state;
        if ((buf == NULL) || (buflen < sizeof(HEADER)))
                return(-1);
        memset(buf, 0, sizeof(HEADER));                                     80
        hp = (HEADER *) buf;
        hp->id = htons(++_rs->id);
        hp->opcode = op;
        hp->pr = (_rs->options & HES_PRIMARY) != 0;
        hp->rd = (_rs->options & RES_RECURSE) != 0;
        hp->rcode = NOERROR;
        cp = buf + sizeof(HEADER);
        buflen -= sizeof(HEADER);
        dpp = dnptrs;
        *dpp++ = buf;                                                       90
        *dpp++ = NULL;
        lastdnptr = dnptrs + sizeof(dnptrs)/sizeof(dnptrs[0]);
        /*
         * perform opcode specific processing
         */
        switch (op) {
         case QUERY:
                if ((buflen -= QFIXEDSZ) < 0)
                        return(-1);
                if ((n = dn_comp((u_char *)dname, (u_char *)cp, buflen,     100
                                        (u_char **)dnptrs, (u_char **)lastdnptr)) < 0)
                        return (-1);
                cp += n;
                buflen -= n;
                _putshort(type, (u_char *)cp);
                cp += sizeof(u_short);
                _putshort(class, (u_char *)cp);
                cp += sizeof(u_short);
                hp->qdcount = htons(1);
                if (op == QUERY || data == NULL)                            110
                        break;
```

```
                /*
                 * Make an additional record for completion domain.
                 */
                buflen -= RRFIXEDSZ;
                if ((n = dn_comp((u_char *)data, (u_char *)cp, buflen,
                                        (u_char **)dnptrs, (u_char **)lastdnptr)) < 0)
                        return (-1);
                cp += n;
                buflen -= n;                                                    120
                _putshort(T_NULL, (u_char *)cp);
                cp += sizeof(u_short);
                _putshort(class, (u_char *)cp);
                cp += sizeof(u_short);
                _putlong(0, (u_char *)cp);
                cp += sizeof(int);
                _putshort(0, (u_char *)cp);
                cp += sizeof(u_short);
                hp->arcount = htons(1);
                break;                                                          130

        case IQUERY:
                /*
                 * Initialize answer section
                 */
                if (buflen < 1 + RRFIXEDSZ + datalen)
                        return (-1);
                *cp++ = '\0';                   /* no domain name */
                _putshort(type, (u_char *)cp);
                cp += sizeof(u_short);                                          140
                _putshort(class, (u_char *)cp);
                cp += sizeof(u_short);
                _putlong(0, (u_char *)cp);
                cp += sizeof(int);
                _putshort(datalen, (u_char *)cp);
                cp += sizeof(u_short);
                if (datalen) {
                        memcpy(cp, data, datalen);
                        cp += datalen;
                }                                                               150
                hp->ancount = htons(1);
                break;

#ifdef ALLOW_UPDATES
                /*
                 * For UPDATEM/UPDATEMA, do UPDATED/UPDATEDA followed by UPDATEA
                 * (Record to be modified is followed by its replacement in msg.)
                 */
        case UPDATEM:
        case UPDATEMA:                                                          160

        case UPDATED:
                /*
                 * The res code for UPDATED and UPDATEDA is the same; user
                 * calls them differently: specifies data for UPDATED; server
```

```
                * ignores data if specified for UPDATEDA.
                */
        case UPDATEDA:
                buflen -= RRFIXEDSZ + datalen;
                if ((n = dn_comp(dname, cp, buflen, dnptrs, lastdnptr)) < 0)
                        return (-1);
                cp += n;
                _putshort(type, cp);
                cp += sizeof(u_short);
                _putshort(class, cp);
                cp += sizeof(u_short);
                _putlong(0, cp);
                cp += sizeof(int);
                _putshort(datalen, cp);
                cp += sizeof(u_short);
                if (datalen) {
                        memcpy(cp, data, datalen);
                        cp += datalen;
                }
                if ( (op == UPDATED) || (op == UPDATEDA) ) {
                        hp->ancount = htons(0);
                        break;
                }
                /* Else UPDATEM/UPDATEMA, so drop into code for UPDATEA */
        case UPDATEA:                           /* Add new resource record */
                buflen -= RRFIXEDSZ + datalen;
                if ((n = dn_comp(dname, cp, buflen, dnptrs, lastdnptr)) < 0)
                        return (-1);
                cp += n;
                _putshort(newrr->r_type, cp);
                cp += sizeof(u_short);
                _putshort(newrr->r_class, cp);
                cp += sizeof(u_short);
                _putlong(0, cp);
                cp += sizeof(int);
                _putshort(newrr->r_size, cp);
                cp += sizeof(u_short);
                if (newrr->r_size) {
                        memcpy(cp, newrr->r_data, newrr->r_size);
                        cp += newrr->r_size;
                }
                hp->ancount = htons(0);
                break;
#endif /* ALLOW_UPDATES */
        }
        return (cp - buf);
}
```

## A.1.10 hes_send.c

```
/*
 * Copyright (c) 1985, 1988 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *        notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *        notice, this list of conditions and the following disclaimer in the
 *        documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *        must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *        may be used to endorse or promote products derived from this software
 *        without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#if defined(LIBC_SCCS) && !defined(lint)
/*static char *sccsid = "from: @(#)hes_send.c        6.45 (Berkeley) 2/24/91";*/
static char *rcsid = "$Id:  hes_send.c,v 1.7 1995/03/29 06:49:51 ghudson Exp $";
#endif /* LIBC_SCCS and not lint */

#include <pthread.h>
#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <time.h>
#include <stropts.h>
#include <poll.h>
#include <sys/socket.h>
#include <sys/uio.h>
#include "hes_internal.h"

enum { SEND_GIVE_UP = -1, SEND_TRY_NEXT = -2, SEND_TRY_SAME = -3,
                SEND_TIMEOUT, SEND_TRUNCATED = -4 };

static int send_datagram(int server, int sock, const char *buf, int buflen,
                                                char *answer, int anslen, int try,
```

```
                                                    struct hes_data *data);
static int send_circuit(int server, const char *buf, int buflen, char *answer,
                                                    int anslen, struct hes_data *data);
static int close_save_errno(int sock);

int hes_send(const char *buf, int buflen, char *answer, int anslen)                    60
{
        struct hes_data *data;
        struct sockaddr_in local;
        int use_virtual_circuit, result, udp_sock, have_seen_same, terrno = 0;
        int try, server;

        data = _hes_init();
        if (!data)
                return -1;
                                                                                       70
        try = 0;
        server = 0;

        /* Try doing connectionless queries if appropriate. */
        if (!(data->state.options & HES_USEVC) && buflen <= PACKETSZ) {
                /* Create and bind a local UDP socket. */
                udp_sock = socket(AF_INET, SOCK_DGRAM, 0);
                if (udp_sock < 0)
                        return -1;
                local.sin_family = AF_INET;                                            80
                local.sin_addr.s_addr = htonl(INADDR_ANY);
                local.sin_port = htons(0);
                if (bind(udp_sock, (struct sockaddr *) &local, sizeof(local)) < 0) {
                        close(udp_sock);
                        return -1;
                }

                /* Cycle through the retries and servers, sending off queries and
                 * waiting for responses. */
                for (; try < data->state.retry; try++) {                               90
                        for (; server < data->state.nscount; server++) {
                                result = send_datagram(server, udp_sock, buf, buflen, answer,
                                                                        anslen, try, data);
                                if (result == SEND_TIMEOUT)
                                        terrno = ETIMEDOUT;
                                else if (result != SEND_TRY_NEXT)
                                        break;
                        }
                        if (server < data->state.nscount)
                                break;                                                 100
                }

                close(udp_sock);
                if (result < 0)
                        errno = (terrno == ETIMEDOUT) ? ETIMEDOUT : ECONNREFUSED;
                else
                        errno = 0;
                if (result != SEND_TRUNCATED)
```

```
                return (result >= 0) ? result : -1;
        }                                                                    110

        /* Either we have to use the virtual circuit, or the server couldn't
         * fit its response in a UDP packet.  Cycle through the retries and
         * servers, sending off queries and waiting for responses.          Allow a
         * response of SEND_TRY_SAME to cause an extra retry once. */
        for (; try < data->state.retry; try++) {
                for (; server < data->state.nscount; server++) {
                        result = send_circuit(server, buf, buflen, answer, anslen, data);
                        terrno = errno;
                        if (result == SEND_TRY_SAME) {                       120
                                if (!have_seen_same)
                                        server--;
                                have_seen_same = 1;
                        } else if (result != SEND_TRY_NEXT) {
                                break;
                        }
                }
        }

        errno = terrno;                                                      130
        return (result >= 0) ? result : -1;
}

static int send_datagram(int server, int sock, const char *buf, int buflen,
                                                char *answer, int anslen, int try,
                                                struct hes_data *data)
{
        int count, timeout;
        struct sockaddr_in local_addr;
        HEADER *request = (HEADER *) buf, *response = (HEADER *) answer;     140
        struct pollfd fd;

#ifdef DEBUG_RESOLVER
        if (_res.options & HES_DEBUG) {
          printf("hes_send:  request:\n");
          _p_query(buf);
        }
#endif /* DEBUG_RESOLVER */
        /* Send a packet to the server. */
        count = sendto(sock, buf, buflen, 0,                                 150
                                (struct sockaddr *) &data->state.nsaddr_list[server],
                                sizeof(struct sockaddr_in));

        if (count != buflen) {
#ifdef DEBUG_RESOLVER
            if (count < 0){
                if (_res.options & HES_DEBUG)
                        perror("send_datagram:sendto");
                }
#endif /* DEBUG_RESOLVER */                                                  160
                return SEND_TRY_NEXT;
        }
```

```
        /* Await a reply with the correct ID. */
        while (1) {
                struct sockaddr_in from;
                int from_len;

                from_len = sizeof(from);
                timeout = 1000 * data->state.retrans << try;                    170
                if (try > 0)
                        timeout /= data->state.nscount;
                fd.fd = sock;
                fd.events = POLLIN;
                count = poll(&fd, 1, timeout);
                if (count <= 0)
                        return SEND_TRY_NEXT;
                count = recvfrom(sock, answer, anslen, 0, (struct sockaddr *) &from,
                                                        &from_len);
                if (count <= 0)                                                 180
                        return SEND_TRY_NEXT;
                /* If the ID is wrong, it's from an old query; ignore it. */
                if (response->id == request->id)
                        break;
#ifdef DEBUG_RESOLVER
            if (_res.options & HES_DEBUG) {
                  printf("hes_sendto:  count=%d, response:\n", count);
                  _p_query(answer);
                }
#endif /* DEBUG_RESOLVER */                                                     190
        }

        /* Report a truncated response unless HES_IGNTC is set.      This will
         * cause the hes_send() loop to fall back to TCP. */
        if (response->tc && !(data->state.options & HES_IGNTC))
                return SEND_TRUNCATED;

        return count;
}
                                                                                200
static int send_circuit(int server, const char *buf, int buflen, char *answer,
                                                        int anslen, struct hes_data *data)
{
        HEADER *response = (HEADER *) answer;
        int sock = -1, result, n, response_len, count;
        unsigned short len;
        struct iovec iov[2];
        char *p, junk[512];

        /* If data->sock is valid, then it's an open connection to the             210
         * first server.  Grab it if it's appropriate; close it if not. */
        if (data->sock) {
                if (server == 0)
                        sock = data->sock;
                else
                        close(data->sock);
```

```
        data->sock = -1;
}

/* Initialize our socket if we didn't grab it from data. */                    220
if (sock == -1) {
        sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0)
                return SEND_GIVE_UP;
        result = connect(sock,
                                        (struct sockaddr *) &data->state.nsaddr_list[server],
                                        sizeof(struct sockaddr_in));
        if (result < 0) {
                close_save_errno(sock);
                return SEND_TRY_NEXT;                                          230
        }
}

/* Send length and message. */
len = htons((unsigned short) buflen);
iov[0].iov_base = (caddr_t) &len;
iov[0].iov_len = sizeof(len);
iov[1].iov_base = (char *) buf;
iov[1].iov_len = buflen;
if (writev(sock, iov, 2) != sizeof(len) + buflen) {                            240
        close_save_errno(sock);
        return SEND_TRY_NEXT;
}

/* Receive length. */
p = (char *) &len;
n = sizeof(len);
while (n) {
        count = read(sock, p, n);
        if (count <= 0) {                                                      250
                /* If we got ECONNRESET, the remote server may have restarted,
                 * and we report SEND_TRY_SAME.  (The main loop will only
                 * allow one of these, so we don't have to worry about looping
                 * indefinitely.) */
                close_save_errno(sock);
                return (errno == ECONNRESET) ? SEND_TRY_SAME : SEND_TRY_NEXT;
        }
        p += count;
        n -= count;
}                                                                              260
len = ntohs(len);
response_len = (len > anslen) ? anslen : len;
len -= response_len;

/* Receive message. */
p = answer;
n = response_len;
while (n) {
        count = read(sock, p, n);
        if (count <= 0) {                                                      270
```

```
                        close_save_errno(sock);
                        return SEND_TRY_NEXT;
                }
                p += count;
                n -= count;
        }

        /* If the reply is longer than our answer buffer, set the truncated
         * bit and flush the rest of the reply, to keep the connection in
         * sync. */                                                          280
        if (len) {
                response->tc = 1;
                while (len) {
                        n = (len > sizeof(junk)) ? sizeof(junk) : len;
                        count = read(sock, junk, n);
                        if (count <= 0) {
                                close_save_errno(sock);
                                return response_len;
                        }
                        len -= count;                                            290
                }
        }

        /* If this is the first server, and HES_USEVC and HES_STAYOPEN are
         * both set, save the connection.  Otherwise, close it. */
        if (server == 0 && (data->state.options & HES_USEVC &&
                                                data->state.options & HES_STAYOPEN))
                data->sock = sock;
        else
                close_save_errno(sock);                                          300

        return response_len;
}

static int close_save_errno(int sock)
{
        int terrno;

        terrno = errno;
        close(sock);                                                             310
        errno = terrno;
}
```

## A.1.11 hesinfo.c

```
/* This is the source code for the hesinfo program, used to test the
 * Hesiod name server.
 *
 *      $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hesinfo.c,v $
 *      $Author: ghudson $
 *      $Athena: hesinfo.c,v 1.4 88/08/07 21:52:19 treese Locked $
 *      $Log: hesinfo.c,v $
 * Revision 1.1  1995/03/18  07:04:51  ghudson
```

```
 * Initial revision
 *
 * Revision 1.6  91/01/21  12:35:27  probe
 * PS/2 integration -- added detailed Hesiod errors
 *
 * Revision 1.5  88/08/07  23:16:50  treese
 * Second-public-distribution
 *
 * Revision 1.4  88/08/07  21:52:19  treese
 * First public distribution
 *
 * Revision 1.3  88/06/12  00:52:34  treese
 * Cleaned up to work with Saber.
 * First public distribution.
 *
 * Revision 1.2  88/06/05  19:51:18  treese
 * Cleaned up for public distribution
 *
 *
 * Copyright 1988 by the Massachusetts Institute of Technology.  See the
 * file <mit-copyright.h> for copying and distribution information.
 */

#include "mit-copyright.h"

#ifndef lint
static char rcsid_hesinfo_c[] = "$Header:  /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hesin
#endif

#include <stdio.h>
#include <hesiod.h>

main(argc, argv)
char *argv[];
{
        register char *cp, **cpp;
        char *hes_to_bind(), **hes_resolve();
        int lflag = 0, errflg = 0, bflag = 0;
        extern int optind;
        char *identifier, *type;
        int c;

        while ((c = getopt(argc, argv, "lb")) != EOF) {
                if (c == 'l') lflag = 1;
                else if (c == 'b') bflag = 1;
                else errflg++;
        }
        if (argc - optind != 2 || errflg) {
                fprintf(stderr,"Usage:  %s [-bl] identifier type\n",argv[0]);
                fprintf(stderr,"           -l selects long format\n");
                fprintf(stderr,"           -b also does hes_to_bind conversion\n");
                exit(2);
        }
        identifier = argv[optind];
```

```
        type = argv[optind+1];

        if (bflag) {
                if (lflag)
                        printf("hes_to_bind(%s, %s) expands to\n",
                            identifier, type);
                cp = hes_to_bind(identifier, type);
                if (cp == NULL) {                                        70
                        printf(" error %d\n", hes_error());
                        exit(1);
                }
                printf("%s\n", cp);
                if (lflag) printf("which ");
        }
        if (lflag)
                printf("resolves to\n");

        cpp = hes_resolve(identifier, type);                             80
        if (cpp == NULL) {
                if (lflag) printf("nothing\n");
                switch(hes_error()) {
                case 0:
                        break;
                case HES_ER_NOTFOUND:
                        fputs("Hesiod name not found\n", stderr);
                        break;
                case HES_ER_CONFIG:
                        fputs("Hesiod configuration error\n", stderr);   90
                        break;
                default:
                        fputs("Unknown Hesiod error\n", stderr);
                        break;
                }
        } else {
                while(*cpp) printf("%s\n", *cpp++);
        }
        if (!cpp)
                exit(1);                                                 100
        else
                exit(0);
}
```

## A.1.12 hesiod.c

```
/* This file is part of the Hesiod library.
 *
 *      $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hesiod.c,v $
 *      $Author: ghudson $
 *      $Athena: hesiod.c,v 1.5 88/08/07 22:00:44 treese Locked $
 *      $Log: hesiod.c,v $
 * Revision 1.15  93/10/22  12:02:36  epeisach
 * Under POSIX, include stdlib.h so that calloc/getenv defined
 * properly for the OS.
```

```
* Also, allow header files in current directory to define the
* return type for hes_resolve and _resolve (instead of redefing)
*
* Revision 1.14  93/10/22  08:17:40  probe
* Use memmove [ANSI] instead of bcopy, except on the platforms where we
* don't have memmove.
*
* Revision 1.13  93/10/22  08:16:06  probe
* ANSI says to use strchr, and even the BSD systems have this function.
*
* Revision 1.12  93/10/21  14:35:55  mar
* include string.h instead of strings.h
*
* Revision 1.11  93/06/15  10:26:37  mar
* handle empty LHS
*
* Revision 1.10  93/04/27  14:03:44  vrt
* compatibility index in solaris is braindamaged.
*
* Revision 1.9  90/07/19  09:20:09  epeisach
* Declare that getenv returns a char*
*
* Revision 1.8  90/07/11  16:46:44  probe
* Patches from <mar>
* Support for HES_DOMAIN environment variable added
*
* Revision 1.9  90/07/11  16:41:18  probe
* Patches from <mar>
* Added description about error codes and the HES_DOMAIN environment
* variable
*
* Revision 1.7  89/11/16  06:49:31  probe
* Uses T_TXT, as defined in the RFC.
*
* Revision 1.6.1.1  89/11/03  17:50:12  probe
* Changes T_TXT to T_UNSPECA.
*
* The BIND 4.8.1 implementation of T_TXT is incorrect; BIND 4.8.1 declares
* it as a NULL terminated string.  The RFC defines T_TXT to be a length
* byte followed by arbitrary changes.
*
* Because of this incorrect declaration in BIND 4.8.1, when this bug is fixed,
* T_TXT requests between machines running different versions of BIND will
* not be compatible (nor is there any way of adding compatibility).
*
* Revision 1.6  88/08/07  23:17:03  treese
* Second-public-distribution
*
* Revision 1.5  88/08/07  22:00:44  treese
* Changed T_UNSPECA to T_TXT.
* Changed C_HESIOD to C_HS.
* Deleted ifdef's based on USE_HS_QUERY -- they're obsolete.
*
* Revision 1.4  88/08/07  21:52:31  treese
```

```
 * First public distribution
 *
 * Revision 1.3  88/06/12  00:52:58  treese
 * Cleaned up to work with Saber.
 * First public distribution.
 *
 * Revision 1.2  88/06/11  22:36:38  treese
 * Cleaned up for public distribution.
 *
 *
 *
 * Copyright 1988 by the Massachusetts Institute of Technology.  See the
 * file <mit-copyright.h> for copying and distribution information.
 */

#include "mit-copyright.h"

#ifndef lint
static char rcsid_hesiod_c[] = "$Header:  /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/h
#endif

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include <netdb.h>
#include <pwd.h>
#ifdef POSIX
#include <stdlib.h>
#else
extern char *malloc(), *calloc(), *getenv();
#endif
#include "resscan.h"
#include "hesiod.h"

#if defined(vax)
#define memcpy(a, b, c) bcopy(b, a, c)
#endif

#define USE_HS_QUERY /* undefine this if your higher-level name servers */
                     /* don't know class HS */

retransXretry_t NoRetryTime = { 0, 0};

char *HesConfigFile = HESIOD_CONF;
static char *Hes_LHS;
static char *Hes_RHS;
int Hes_Errno = HES_ER_UNINIT;

static void _hes_init();
```

```
int
hes_init()
{                                                                    120
        register FILE *fp;
        register char *key, *cp, **cpp;
        int len;
        char buf[MAXDNAME+7];

        if (Hes_Errno != HES_ER_UNINIT && Hes_Errno != HES_ER_CONFIG)
                return 0;
#ifndef INTERNAL_RESOLVER
        res_init();
#endif                                                               130
        Hes_Errno = HES_ER_UNINIT;
        Hes_LHS = NULL; Hes_RHS = NULL;

        if ((fp = fopen(HesConfigFile, "r")) == NULL) {
                /* use defaults compiled in */
                /* no file or no access uses defaults */
                /* but poorly formed file returns error */
                Hes_LHS = DEF_LHS; Hes_RHS = DEF_RHS;
        } else {
                while(fgets(buf, MAXDNAME+7, fp) != NULL) {          140
                        cp = buf;
                        if (*cp == '#' || *cp == '\n') continue;
                        while(*cp == ' ' || *cp == '\t') cp++;
                        key = cp;
                        while(*cp != ' ' && *cp != '\t' && *cp != '=') cp++;
                        *cp++ = '\0';
                        if (strcmp(key, "lhs") == 0) cpp = &Hes_LHS;
                        else if (strcmp(key, "rhs") == 0) cpp = &Hes_RHS;
                        else continue;
                        while(*cp == ' ' || *cp == '\t' || *cp == '=') cp++;   150
                        if (*cp != '.' && *cp != '\n') {
                                Hes_Errno = HES_ER_CONFIG;
                                fclose(fp);
                        }
                        len = strlen(cp);
                        *cpp = calloc((unsigned int) len, sizeof(char));
                        (void) strncpy(*cpp, cp, len-1);
                }
                fclose(fp);
        }                                                            160
        /* see if the RHS is overridden by environment variable */
        if ((cp = getenv("HES_DOMAIN")) != NULL)
                Hes_RHS = strcpy(malloc(strlen(cp)+1),cp);
        /* the LHS may be null, the RHS must not be null */
        if (Hes_RHS == NULL)
                Hes_Errno = HES_ER_CONFIG;
        else
                Hes_Errno = HES_ER_OK;
        return(Hes_Errno);
}                                                                    170
```

```
char *
hes_to_bind(HesiodName, HesiodNameType)
char *HesiodName, *HesiodNameType;
{
        static char buffer[MAXDNAME];
        int status;

        if (Hes_Errno == HES_ER_UNINIT || Hes_Errno == HES_ER_CONFIG)
                (void) hes_init();                                          180
        if (Hes_Errno == HES_ER_CONFIG) return(NULL);
        status = hes_to_bind_r(HesiodName, HesiodNameType, buffer, MAXDNAME);
        if (status == HES_ER_OK) {
                return(buffer);
        } else {
                Hes_Errno = status;
                return(NULL);
        }
}
                                                                            190
int
hes_to_bind_r(HesiodName, HesiodNameType, buffer, bufsize)
char *HesiodName, *HesiodNameType, *buffer;
int bufsize;
{
        register char *cp;
        char *cpp[2];
        char *RHS;
        int len;
                                                                            200
        *cpp = NULL;
        if (Hes_Errno == HES_ER_CONFIG)
                return(HES_ER_CONFIG);
        if (Hes_Errno == HES_ER_UNINIT) {
                fprintf(stderr, "Library not initialized in hes_to_bind_r().\n");
                abort();
        }
        if (cp = strchr(HesiodName,'@')) {
                if (cp - HesiodName > bufsize - 1)
                        return(HES_ER_RANGE);                               210
                if (strchr(cp + 1,'.'))
                        RHS = cp + 1;
                else
                        if (hes_resolve_r(cp + 1, "rhs-extension", cpp, 2) == HES_ER_OK
                                && *cpp != NULL)
                                RHS = *cpp;
                        else {
                                return(HES_ER_NOTFOUND);
                        }
                (void) strncpy(buffer,HesiodName,cp - HesiodName);          220
                buffer[cp - HesiodName] = '\0';
        } else {
                RHS = Hes_RHS;
                if ((int) strlen(HesiodName) > bufsize - 1)
                        return(HES_ER_RANGE);
```

```
                (void) strcpy(buffer, HesiodName);
        }
        len = strlen(buffer) + 1 + strlen(HesiodNameType);
        if (Hes_LHS)
                len += strlen(Hes_LHS) + ((Hes_LHS[0] != '.') ? 1 : 0);
        len += strlen(RHS) + ((RHS[0] != '.') ? 1 : 0);
        if (len > bufsize - 1) {
                if (*cpp)
                        free(*cpp);
                return(HES_ER_RANGE);
        }
        (void) strcat(buffer, ".");
        (void) strcat(buffer, HesiodNameType);
        if (Hes_LHS) {
                if (Hes_LHS[0] != '.')
                        (void) strcat(buffer,".");
                (void) strcat(buffer, Hes_LHS);
        }
        if (RHS[0] != '.')
                (void) strcat(buffer,".");
        (void) strcat(buffer, RHS);
        if (*cpp)
                free(*cpp);
        return(HES_ER_OK);
}

char **
hes_resolve(HesiodName, HesiodNameType)
char *HesiodName, *HesiodNameType;
{
        static char *retvec[100];
        int status;

        if (Hes_Errno == HES_ER_UNINIT || Hes_Errno == HES_ER_CONFIG)
                (void) hes_init();
        if (Hes_Errno == HES_ER_CONFIG) return(NULL);
        status = hes_resolve_r(HesiodName, HesiodNameType, retvec, 100);
        if (status == HES_ER_OK) {
                return(retvec);
        } else {
                Hes_Errno = status;
                return(NULL);
        }
}

int
hes_resolve_r(HesiodName, HesiodNameType, retvec, retveclen)
char *HesiodName, *HesiodNameType, **retvec;
int retveclen;
{
        register char *cp;
        char buffer[MAXDNAME], nmsgbuf[NMSGSIZE], databuf[DATASIZE];
        char *ocp, *dst;
        int i, j, n;
```

```
        struct nsmsg *ns;                                                    280
        rr_t *rp;
        extern int errno;

        if (Hes_Errno == HES_ER_CONFIG)
                return(HES_ER_CONFIG);
        if (Hes_Errno == HES_ER_UNINIT) {
                fprintf(stderr, "Library not initialized in hes_resolve_r().\n");
                abort();
        }
        i = hes_to_bind_r(HesiodName, HesiodNameType, buffer, MAXDNAME);     290
        if (i != HES_ER_OK) return(i);
        cp = buffer;
        errno = 0;
        ns = _resolve(cp, C_HS, T_TXT, NoRetryTime, nmsgbuf, databuf);
        if (ns == NULL && (errno == ETIMEDOUT || errno == ECONNREFUSED))
                return(HES_ER_NET);
        if (ns == NULL || ns->ns_off <= 0)
                return(HES_ER_NOTFOUND);
        for(i = j = 0, rp = &ns->rr; i < ns->ns_off; rp++, i++) {
                if (j >= retveclen) {                                        300
                        for (i = 0; i < j; i++)
                                free(retvec[i]);
                        return(HES_ER_RANGE);
                }
                if (
                   rp->class == C_HS &&
                   rp->type == T_TXT) { /* skip CNAME records */
                        retvec[j] = calloc(rp->dlen + 1, sizeof(char));
                        if (retvec[j] == NULL) {
                                for (i = 0; i < j; i++)                      310
                                        free(retvec[i]);
                                return(HES_ER_NOMEM);
                        }
                        dst = retvec[j];
                        ocp = cp = rp->data;
                        while (cp < ocp + rp->dlen) {
                           n = (unsigned char) *cp++;
                           (void) memcpy(dst, cp, n);
                           cp += n;
                           dst += n;                                         320
                        }
                        *dst = 0;
                        j++;
                }
        }
        if (j >= retveclen) {
                for (i = 0; i < j; i++)
                        free(retvec[i]);
                return(HES_ER_RANGE);
        }                                                                    330
        retvec[j] = 0;
        return(HES_ER_OK);
}
```

```
int
hes_error()
{
        return Hes_Errno;
}
```

## A.1.13 hesmailhost.c

```
/* This file contains hes_postoffice, which retrieves post-office information
 * for a user.
 *
 * For copying and distribution information, see the file <mit-copyright.h>
 *
 * Original version by Steve Dyer, IBM/Project Athena.
 *
 *      $Author: ghudson $
 *      $Athena: hesmailhost.c,v 1.4 88/08/07 21:52:45 treese Locked $
 *      $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/code/RCS/hesmailhost.c,v$
 *      $Log: hesmailhost.c,v $
 * Revision 1.1  1995/03/18  07:04:51  ghudson
 * Initial revision
 *
 * Revision 1.6  93/10/21  14:36:09  mar
 * include string.h instead of strings.h
 *
 * Revision 1.5  88/08/07  23:17:10  treese
 * Second-public-distribution
 *
 * Revision 1.4  88/08/07  21:52:45  treese
 * First public distribution
 *
 * Revision 1.3  88/06/12  00:53:06  treese
 * Cleaned up to work with Saber.
 * First public distribution.
 *
 * Revision 1.2  88/06/05  19:51:36  treese
 * Cleaned up for public distribution
 *
 *
 */

#include "mit-copyright.h"

#ifndef lint
static char rcsid_hesmailhost_c[] = "$Header:  /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/code/
#endif

#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <netdb.h>
```

```
#include <pwd.h>

#include "hesiod.h"

#define LINESIZE 80

extern int Hes_Errno;                                                    50

struct hes_postoffice *
hes_getmailhost(user)
char *user;
{
        static struct hes_postoffice ret;
        static char linebuf[LINESIZE];
        int status;

        hes_init();                                                      60
        status = hes_getmailhost_r(user, &ret, linebuf, LINESIZE);
        if (status == HES_ER_OK) {
                return(&ret);
        } else {
                Hes_Errno = status;
                return(NULL);
        }
}

int                                                                      70
hes_getmailhost_r(user, ret, linebuf, bufsize)
char *user, *linebuf;
struct hes_postoffice *ret;
int bufsize;
{
        char *p, *cp[2];
        int status;

        status = hes_resolve_r(user, "pobox", cp, 2);
        if (status != HES_ER_OK)                                         80
                return(status);
        if (*cp == NULL)
                return(HES_ER_INVAL);
        if ((int) strlen(*cp) > bufsize - 1) {
                free(*cp);
                return(HES_ER_RANGE);
        }
        strcpy(linebuf, *cp);
        free(*cp);
        ret->po_type = linebuf;                                          90
        p = linebuf;
        while(!isspace(*p)) p++;
        *p++ = '\0';
        ret->po_host = p;
        while(!isspace(*p)) p++;
        *p++ = '\0';
        ret->po_name = p;
```

```
        return(HES_ER_OK);
}
```

## A.1.14 hespwnam.c

```
/* This file contains hes_getpwnam, for retrieving passwd information about
 * a user.
 *
 * For copying and distribution information, see the file <mit-copyright.h>
 *
 * Original version by Steve Dyer, IBM/Project Athena.
 *
 *      $Author: ghudson $
 *      $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hespwnam.c,v $
 *      $Athena: hespwnam.c,v 1.4 88/08/07 21:52:51 treese Locked $                    10
 */

#include "mit-copyright.h"

#ifndef lint
static char rcsid_pwnam_c[] = "$Header:  /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hespw
#endif
#include <stdio.h>
#include <pwd.h>
#include <string.h>                                                                    20
#include <netdb.h>
#include "hesiod.h"

static struct passwd pw_entry;
static char buf[256];

static char *_NextPWField();    /* For later definition in file */

extern int Hes_Errno;

                                                                                       30
static int
hes_getpwcommon(arg, which, entry, buf, bufsize)
        char *arg;
        int which;          /* 0=hes_getpwnam, 1=hes_getpwuid */
        struct passwd *entry;
        char *buf;
        int bufsize;
{
        register char *p;
        char *pp[2];                                                                   40
        int status;

        status = hes_resolve_r(arg, which ? "uid" : "passwd", pp, 2);
        if (status != HES_ER_OK)
                return(status);
        if (*pp == NULL)
                return(HES_ER_INVAL);
        /* choose only the first response (only 1 expected) */
```

```
        if ((int) strlen(pp[0]) > bufsize - 1) {
                free(pp[0]);                                              50
                return(HES_ER_RANGE);
        }
        (void) strcpy(buf, pp[0]);
        free(pp[0]);
        p = buf;
        entry->pw_name = p;
        p = _NextPWField(p);
        entry->pw_passwd = p;
        p = _NextPWField(p);
        entry->pw_uid = atoi(p);                                          60
        p = _NextPWField(p);
        entry->pw_gid = atoi(p);
#if defined(_AIX) && (AIXV < 31)
        entry->pw_quota = 0;
        entry->pw_age =
        entry->pw_comment = "";
#endif
        p = _NextPWField(p);
        entry->pw_gecos = p;
        p = _NextPWField(p);                                              70
        entry->pw_dir = p;
        p = _NextPWField(p);
        entry->pw_shell = p;
        while (*p && *p != '\n')
                p++;
        *p = '\0';
        return(HES_ER_OK);
}

/* Move the pointer forward to the next colon-separated field in the     80
 * password entry.
 */

static char *
_NextPWField(ptr)
char *ptr;
{
        while (*ptr && *ptr != '\n' && *ptr != ':')
                ptr++;
        if (*ptr)                                                         90
                *ptr++ = '\0';
        return(ptr);
}

struct passwd *
hes_getpwnam(nam)
        char *nam;
{
        int status;
                                                                          100
        hes_init();
        status = hes_getpwcommon(nam, 0, &pw_entry, buf, sizeof(buf));
```

```
        if (status == HES_ER_OK) {
                return(&pw_entry);
        } else {
                Hes_Errno = status;
                return(NULL);
        }
}

struct passwd *
hes_getpwuid(uid)
        int uid;
{
        char uidstr[16];
        int status;

        hes_init();
        sprintf(uidstr, "%d", uid);
        status = hes_getpwcommon(uidstr, 1, &pw_entry, buf, sizeof(buf));
        if (status == HES_ER_OK) {
                return(&pw_entry);
        } else {
                Hes_Errno = status;
                return(NULL);
        }
}

int
hes_getpwnam_r(nam, entry, buf, bufsize)
        char *nam;
        struct passwd *entry;
        char *buf;
        int bufsize;
{
        return hes_getpwcommon(nam, 0, entry, buf, bufsize);
}

int
hes_getpwuid_r(uid, entry, buf, bufsize)
        int uid;
        struct passwd *entry;
        char *buf;
        int bufsize;
{
        char uidstr[16];

        sprintf(uidstr, "%d", uid);
        return hes_getpwcommon(uidstr, 1, entry, buf, bufsize);
}
```

## A.1.15 hesservbyname.c

```
/*
```

```
 * Copyright (c) 1983 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

#if defined(LIBC_SCCS) && !defined(lint)
static char sccsid[] = "@(#)getservbyname.c	5.3 (Berkeley) 5/19/86";
#endif LIBC_SCCS and not lint

#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <ctype.h>
#include <pwd.h>
#ifdef POSIX
#include <stdlib.h>
#else
extern char *malloc();
#endif
#include "hesiod.h"

extern int Hes_Errno;

struct servent *
hes_getservbyname(name, proto)
        char *name, *proto;
{
        static struct servent result;
        static char buffer[2048];
        int status;

        hes_init();
        status = hes_getservbyname_r(name, proto, &result, buffer, 2048);
        if (status == HES_ER_OK) {
           return(&result);
        } else {
           Hes_Errno = status;
           return(NULL);
        }
}

int
hes_getservbyname_r(name, proto, result, buffer, buflen)
        char *name, *proto, *buffer;
        struct servent *result;
        int buflen;
{
        register char **cp, **aliases = (char **) buffer;
        char *buf[100], *line;
        register int i = 0;
        int status;

        status = hes_resolve_r(name, "service", buf, 100);
```

```
if (status != HES_ER_OK) return(status);
status = HES_ER_NOTFOUND;
for (cp = buf; *cp; cp++) {
        register char *servicename, *protoname, *port, *l = *cp;
        register int len = strlen(l);                                    60

        /* Find the service name. */
        while(*l && (*l == ' ' || *l == '\t'))
                l++;
        servicename = l;
        while(*l && *l != ' ' && *l != '\t' && *l != ';')
                l++;
        if (*l == '\0') /* malformed entry */
                continue;
        *l++ = '\0';                                                     70

        /* Find the protocol name and check it. */
        while(*l && (*l == ' ' || *l == '\t'))
                l++;
        protoname = l;
        while(*l && *l != ' ' && *l != ';')
                l++;
        if (*l == '\0') /* malformed entry */
                continue;
        *l++ = '\0';                                                     80
        if (cistrcmp(proto, protoname)) /* wrong protocol */
                continue;

        /* Find the port number. */
        while(*l && (*l == ' ' || *l == '\t' || *l == ';'))
                l++;
        if (*l == '\0') /* malformed entry */
                continue;
        port = l;
                                                                         90
        while(*l && *l != ' ' && *l != '\t' && *l != ';')
                l++;
        if (*l)
                *l++ = '\0';
        while (*l) {
                if ((i + 2) * sizeof(char *) + len >= buflen)
                        break;
                aliases[i++] = l;
                while(*l && !isspace(*l))
                        l++;                                             100
                if (*l)
                        *l++ = 0;
        }
        if ((i + 1) * sizeof(char *) + len >= buflen) {
                status = HES_ER_RANGE;
                break;
        }
        aliases[i++] = 0;
        line = (char *)(aliases + i);
```

```
                memcpy(line, *cp, len + 1);
                result->s_name = line;
                result->s_port = htons((unsigned short) atoi(port));
                result->s_proto = line + (protoname - *cp);
                result->s_aliases = aliases;
                for (; *aliases; aliases++)
                        *aliases = line + (*aliases - *cp);
                status = HES_ER_OK;
                break;
        }
        for (cp = buf; *cp; cp++)
                free(*cp);
        return(status);
}
```

## A.1.16 hestest.c

```
/* This file is part of the Hesiod library.
 *
 *      $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hesiod.c,v $
 *      $Author: ghudson $
 *      $Athena: hestest.c,v 1.5 88/08/07 22:00:44 treese Locked $
 *      $Log: hestest.c,v $
 * Revision 1.1  1995/03/18  07:04:51  ghudson
 * Initial revision
 *
 *
 * Copyright 1988 by the Massachusetts Institute of Technology.  See the
 * file <mit-copyright.h> for copying and distribution information.
 */

#include "mit-copyright.h"

#ifndef lint
static char rcsid_hestest_c[] = "$Header:  /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/
#endif

#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <pwd.h>
#include <netinet/in.h>
#include "hesiod.h"

char *word_end(char *s)
{
   while (*s && !isspace(*s))
        s++;
   return s;
}

char *find_word(char *s)
{
```

```
    while (isspace(*s))
        s++;
    return s;
}

char *get_word(char *p, char *buf)
{
    char *q = word_end(p);

    strncpy(buf, p, q - p);
    buf[q - p] = 0;
    return q;
}

char *get_field(char *p, int delim, char *buf)
{
    char *q = strchr(p, delim);

    if (q) {
        strncpy(buf, p, q - p);
        buf[q - p] = 0;
        return q + 1;
    } else {
        strcpy(buf, p);
        return NULL;
    }
}

int compare_vector(char **vector, char *spec)
{
    char field[100];

    for (; *vector; vector++) {
        spec = get_field(spec, '\\', field);
        if ((!spec && vector[1]) || strcmp(*vector, field) != 0)
            return -1;
    }
    return (spec) ? -1 : 0;
}

int compare_pwnam(struct passwd *pw, char *spec)
{
    char field[100];

    spec = get_field(spec, ':', field);
    if (!spec || strcmp(pw->pw_name, field) != 0)
        return -1;
    spec = get_field(spec, ':', field);
    if (!spec || strcmp(pw->pw_passwd, field) != 0)
        return -1;
    spec = get_field(spec, ':', field);
    if (pw->pw_uid != atoi(field))
        return -1;
    spec = get_field(spec, ':', field);
```

```
    if (pw->pw_gid != atoi(field))
        return -1;
    spec = get_field(spec, ':', field);
    if (!spec || strcmp(pw->pw_gecos, field) != 0)
        return -1;
    spec = get_field(spec, ':', field);
    if (!spec || strcmp(pw->pw_dir, field) != 0)
        return -1;
    spec = get_field(spec, ':', field);
    if (spec || strcmp(pw->pw_shell, field) != 0)                    100
        return -1;
    return 0;
}

int compare_serv(struct servent *serv, char *spec)
{
    char field[100], **aliases;

    spec = get_field(spec, ':', field);
    if (!spec || strcmp(serv->s_name, field) != 0)                   110
        return -1;
    spec = get_field(spec, ':', field);
    if (!spec || strcmp(serv->s_proto, field) != 0)
        return -1;
    spec = get_field(spec, ':', field);
    if (serv->s_port != htons(atoi(field)))
        return -1;
    for (aliases = serv->s_aliases; *aliases; aliases++) {
        spec = get_field(spec, '\\', field);
        if ((!spec && aliases[1]) || strcmp(*aliases, field) != 0)   120
            return -1;
    }
    return (spec) ? -1 : 0;
}

int compare_office(struct hes_postoffice *office, char *spec)
{
    char field[100];

    spec = get_field(spec, ':', field);                              130
    if (!spec || strcmp(office->po_type, field) != 0)
        return -1;
    spec = get_field(spec, ':', field);
    if (!spec || strcmp(office->po_host, field) != 0)
        return -1;
    spec = get_field(spec, ':', field);
    if (spec || strcmp(office->po_name, field) != 0)
        return -1;
    return 0;
}                                                                    140

void free_ptrs(char **ptrs)
{
    for (; *ptrs; ptrs++)
```

```
        free(*ptrs);
}

int main(argc, argv)
    int argc;
    char **argv;                                                    150
{
    FILE *fp;
    char buf[1000], *p, *q, name[100], type[100], *vec[100], **vecp;
    char buf2[1000];
    int line, status;
    struct passwd pw, *pwp;
    struct servent serv, *servp;
    struct hes_postoffice office, *officep;

    if (argc != 2) {                                                160
        fprintf(stderr, "Usage:  %s filename\n", argv[0]);
        exit(1);
    }

    fp = fopen(argv[1], "r");
    if (!fp) {
        fprintf(stderr, "Couldn't open %s for reading.\n", argv[1]);
        exit(1);
    }
                                                                    170
    hes_init();
    line = 0;
    while (fgets(buf, sizeof(buf), fp)) {
        line++;

        /* Strip off trailing spaces (inefficiently). */
        while (isspace(buf[strlen(buf) - 1]))
            buf[strlen(buf) - 1] = 0;

        /* Get the first word, discard comment lines and invalid lines. */   180
        q = word_end(p = find_word(buf));
        if (*p == '#' || !*p)
            continue;
        if (!*q) {
            fprintf(stderr, "Invalid test:  %s\n", buf);
            continue;
        }

        /* Test for hes_resolve and hes_resolve_r. */
        if (q - p == 7 && strncmp(p, "resolve", 7) == 0) {               190
            q = get_word(find_word(q), name);
            q = get_word(find_word(q), type);
            p = find_word(q);
            status = hes_resolve_r(name, type, vec, 100);
            if (status != HES_ER_OK && !(*p == 'E' && p[1] == 0)) {
                printf("Line %d failed (hes_resolve_r error %d).\n",
                    line, status);
                continue;
```

```
    }
    if (status == HES_ER_OK && compare_vector(vec, p) < 0) {
        printf("Line %d failed in hes_resolve_r().\n", line);
        free_ptrs(vec);
        continue;
    }
    if (status == HES_ER_OK)
        free_ptrs(vec);
    vecp = hes_resolve(name, type);
    if (!vecp) {
        if (*p == 'E' && p[1] == 0) {
            printf("Line %d passed (errors %d, %d).\n",
                line, status, hes_error());
        } else {
            printf("Line %d failed (hes_resolve error %d).\n",
                line, hes_error());
        }
        continue;
    }
    if (compare_vector(vecp, p) < 0)
        printf("Line %d failed in hes_resolve().\n", line);
    else
        printf("Line %d passed.\n", line);
    free_ptrs(vecp);

/* Test for hes_getpwnam and hes_getpwnam_r. */
} else if (q - p == 8 && strncmp(p, "getpwnam", 8) == 0) {
    q = get_word(find_word(q), name);
    p = find_word(q);
    status = hes_getpwnam_r(name, &pw, buf2, sizeof(buf2));
    if (status != HES_ER_OK && !(*p == 'E' && p[1] == 0)) {
        printf("Line %d failed (hes_getpwnam_r error %d).\n",
            line, status);
        continue;
    }
    if (status == HES_ER_OK && compare_pwnam(&pw, p) < 0) {
        printf("Line %d failed in hes_getpwnam_r().\n", line);
        continue;
    }
    pwp = hes_getpwnam(name);
    if (!pwp) {
        if (*p == 'E' && p[1] == 0) {
            printf("Line %d passed (errors %d, %d).\n",
                line, status, hes_error());
        } else {
            printf("Line %d failed (hes_getpwnam error %d).\n",
                line, hes_error());
        }
        continue;
    }
    if (compare_pwnam(pwp, p) < 0)
        printf("Line %d failed in hes_getpwnam().\n", line);
    else
        printf("Line %d passed.\n", line);
```

```
/* Test for hes_getpwuid and hes_getpwuid_r. */
} else if (q - p == 8 && strncmp(p, "getpwuid", 8) == 0) {
    q = get_word(find_word(q), name);
    p = find_word(q);
    status = hes_getpwuid_r(atoi(name), &pw, buf2, sizeof(buf2));
    if (status != HES_ER_OK && !(*p == 'E' && p[1] == 0)) {
            printf("Line %d failed (hes_getpwuid_r error %d).\n",
                line, status);
            continue;
    }
    if (status == HES_ER_OK && compare_pwnam(&pw, p) < 0) {
            printf("Line %d failed in hes_getpwuid_r().\n", line);
            continue;
    }
    pwp = hes_getpwuid(atoi(name));
    if (!pwp) {
            if (*p == 'E' && p[1] == 0) {
                printf("Line %d passed (errors %d, %d).\n",
                        line, status, hes_error());
            } else {
                printf("Line %d failed (hes_getpwuid error %d).\n",
                        line, hes_error());
            }
            continue;
    }
    if (compare_pwnam(pwp, p) < 0)
            printf("Line %d failed in hes_getpwuid().\n", line);
    else
            printf("Line %d passed.\n", line);

/* Test for hes_getservbyname and hes_getservbyname_r. */
} else if (q - p == 13 && strncmp(p, "getservbyname", 13) == 0) {
    q = get_word(find_word(q), name);
    q = get_word(find_word(q), type);
    p = find_word(q);
    status = hes_getservbyname_r(name, type, &serv, buf2,
                                        sizeof(buf2));
    if (status != HES_ER_OK && !(*p == 'E' && p[1] == 0)) {
            printf("Line %d failed (hes_getservbyname_r error %d).\n",
                line, status);
            continue;
    }
    if (status == HES_ER_OK && compare_serv(&serv, p) < 0) {
            printf("Line %d failed in hes_getservbyname_r.\n", line);
            continue;
    }
    servp = hes_getservbyname(name, type);
    if (!servp) {
            if (*p == 'E' && p[1] == 0) {
                printf("Line %d passed (errors %d, %d).\n",
                        line, status, hes_error());
            } else {
                printf("Line %d failed (hes_getservbyname error %d).\n",
```

```
                        line, hes_error());
                }
                continue;
        }
        if (compare_serv(servp, p) < 0)
                printf("Line %d failed in hes_getservbyname().\n", line);
        else
                printf("Line %d passed.\n", line);

    /* Test for hes_getmailhost and hes_getmailhost_r. */
    } else if (q - p == 11 && strncmp(p, "getmailhost", 11) == 0) {
        q = get_word(find_word(q), name);
        p = find_word(q);
        status = hes_getmailhost_r(name, &office, buf2, sizeof(buf2));
        if (status != HES_ER_OK && !(*p == 'E' && p[1] == 0)) {
                printf("Line %d failed (hes_getmailhost_r error %d).\n",
                    line, status);
                continue;
        }
        if (status == HES_ER_OK && compare_office(&office, p) < 0) {
                printf("Line %d failed in hes_getmailhost_r.\n", line);
                continue;
        }
        officep = hes_getmailhost(name);
        if (!officep) {
                if (*p == 'E' && p[1] == 0) {
                    printf("Line %d passed (errors %d, %d).\n",
                            line, status, hes_error());
                } else {
                    printf("Line %d failed (hes_getmailhost error %d).\n",
                            line, hes_error());
                }
                continue;
        }
        if (compare_office(officep, p) < 0)
                printf("Line %d failed in hes_getmailhost().\n", line);
        else
                printf("Line %d passed.\n", line);

    } else {
                printf("Line %d invalid.\n", line);
        }
    }
}
```

## A.1.17 hesthread.c

```
/* This is the source code for the hesinfo program, used to test the
 * Hesiod name server.
 *
 *      $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hesinfo.c,v $
 *      $Author: ghudson $
```

```
 *      $Athena: hesinfo.c,v 1.4 88/08/07 21:52:19 treese Locked $
 *      $Log: hesinfo.c,v $
 * Revision 1.1  1995/03/18  07:04:51  ghudson
 * Initial revision
 *                                                                    10
 * Revision 1.6  91/01/21  12:35:27  probe
 * PS/2 integration - added detailed Hesiod errors
 *
 * Revision 1.5  88/08/07  23:16:50  treese
 * Second-public-distribution
 *
 * Revision 1.4  88/08/07  21:52:19  treese
 * First public distribution
 *
 * Revision 1.3  88/06/12  00:52:34  treese                           20
 * Cleaned up to work with Saber.
 * First public distribution.
 *
 * Revision 1.2  88/06/05  19:51:18  treese
 * Cleaned up for public distribution
 *
 *
 * Copyright 1988 by the Massachusetts Institute of Technology.  See the
 * file <mit-copyright.h> for copying and distribution information.
 */                                                                   30

#include "mit-copyright.h"

#ifndef lint
static char rcsid_hesinfo_c[] = "$Header:  /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hesin
#endif

#include <pthread.h>
#include <stdio.h>
#include <hesiod.h>                                                   40

void *resolve(arg)
void *arg;
{
        char *buf[100], **cpp, **argv = (char **) arg;
        char *name = argv[0], *type = argv[1];

        switch (hes_resolve_r(name, type, buf, 100)) {
                case HES_ER_OK:
                        if (*buf && !buf[1]) {                        50
                                printf("%s %s => %s\n", name, type, *buf);
                        } else {
                                flockfile(stdout);
                                printf("%s %s:\n", name, type);
                                for (cpp = buf; *cpp; cpp++)
                                        puts(*cpp);
                                funlockfile(stdout);
                        }
                        break;
```

```
                case HES_ER_NOTFOUND:                                    60
                        fputs("Hesiod name not found\n", stderr);
                        break;
                case HES_ER_CONFIG:
                        fputs("Hesiod configuration error\n", stderr);
                        break;
                default:
                        fputs("Unknown Hesiod error\n", stderr);
                        break;
        }
        return NULL;                                                     70
}

main(argc, argv)
int argc;
char *argv[];
{
        pthread_t id;

        if (argc % 2 != 1) {
                fprintf(stderr,"Usage:  %s identifier type ...\n", argv[0]);   80
                exit(2);
        }

        hes_init();

        argv++;
        while (*argv) {
                pthread_create(&id, NULL, resolve, argv);
                pthread_detach(id);
                argv += 2;                                               90
        }
        pthread_exit(NULL);
}
```

## A.1.18 resolve.c

```
#ifndef lint
static char *RCS_ID = "$Header:  /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/resol
#endif
/*
 * $Author: ghudson $
 * $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/resolve.c,v $
 * $Athena: resolve.c,v 1.4 88/08/07 21:58:40 treese Locked $
 */

#define _RESOLVE_C_                                                      10

#ifdef SOLARIS
#define res_mkquery hes_mkquery
#define res_send hes_send
#endif
```

```
#include <string.h>
#include <sys/types.h>
#include <sys/param.h>
#include <netinet/in.h>
#include <sys/errno.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include "resscan.h"

extern int errno;

static dn_skip();

static caddr_t
rr_scan(cp, rr)
    char *cp;
    rr_t *rr;
{
    register int n;

    if ((n = dn_skip(cp)) < 0) {
        errno = EINVAL;
        return((char *)NULL);
    }

    cp += n;
    rr->type = _getshort(cp);
    cp += sizeof(u_short/*type*/);

    rr->class = _getshort(cp);
#ifdef _alpha
    cp += sizeof(u_short/*class*/) + sizeof(u_int/*ttl*/);
#else
    cp += sizeof(u_short/*class*/) + sizeof(u_long/*ttl*/);
#endif

    rr->dlen = (int)_getshort(cp);
    rr->data = cp + sizeof(u_short/*dlen*/);

    return(rr->data + rr->dlen);
}

nsmsg_p
res_scan(msg, nmsgbuf, databuf)
    char *msg, *nmsgbuf, *databuf;
{
    register char *cp;
    register rr_t *rp;
    register HEADER *hp;
    register char *data = databuf;
    register int n, n_an, n_ns, n_ar, nrec;
    register nsmsg_t *mess = (nsmsg_t *)nmsgbuf;
```

```
hp = (HEADER *)msg;                                                    70
cp = msg + sizeof(HEADER);
n_an = ntohs(hp->ancount);
n_ns = ntohs(hp->nscount);
n_ar = ntohs(hp->arcount);
nrec = n_an + n_ns + n_ar;

mess->len = 0;
mess->hd = hp;
mess->ns_off = n_an;
mess->ar_off = n_an + n_ns;                                            80
mess->count = nrec;
rp = &mess->rr;

/* skip over questions */
if (n = ntohs(hp->qdcount)) {
    while (--n >= 0) {
        register int i;
        if ((i = dn_skip(cp)) < 0)
            return((nsmsg_t *)NULL);
        cp += i + (sizeof(u_short/*type*/) + sizeof(u_short/*class*/));  90
    }
}

/* scan answers */
if (n = n_an) {
    while (--n >= 0) {
        if ((cp = rr_scan(cp, rp)) == NULL)
            return((nsmsg_t *)NULL);
        (void) strncpy(data, rp->data, rp->dlen);
        rp->data = data;                                               100
        data += rp->dlen;
        *data++ = '\0';
        rp++;
    }
}

/* scan name servers */
if (n = n_ns) {
    while (--n >= 0) {
        if ((cp = rr_scan(cp, rp)) == NULL)                            110
            return((nsmsg_t *)NULL);
        (void) strncpy(data, rp->data, rp->dlen);
        rp->data = data;
        data += rp->dlen;
        *data++ = '\0';
        rp++;
    }
}

/* scan additional records */                                          120
if (n = n_ar) {
    while (--n >= 0) {
        if ((cp = rr_scan(cp, rp)) == NULL)
```

```
            return((nsmsg_t *)NULL);
         (void) strncpy(data, rp->data, rp->dlen);
         rp->data = data;
         data += rp->dlen;
         *data++ = '\0';
         rp++;
      }
   }

   mess->len = (int)cp - (int)msg;

   return(mess);
}

/*
 * Resolve name into data records
 */

nsmsg_p
_resolve(name, class, type, patience, nmsgbuf, databuf)
   char *name, *nmsgbuf, *databuf;
   int class, type;
   retransXretry_t patience;
{
   char qbuf[PACKETSZ], abuf[PACKETSZ];
   register int n;

#ifdef DEBUG
   if (_res.options & RES_DEBUG)
      printf("_resolve:  class = %d, type = %d\n", class, type);
#endif

   if (class < 0 || type < 0) {
      errno = EINVAL;
      return((nsmsg_t *)NULL);
   }

   n = res_mkquery(QUERY, name, class, type, (char *)0, 0, NULL, qbuf, PACKETSZ);
   if (n < 0) {
      errno = EMSGSIZE;
      return((nsmsg_t *)NULL);
   }

   n = res_send(qbuf, n, abuf, PACKETSZ);
   if (n < 0) {
      errno = ECONNREFUSED;
      return((nsmsg_t *)NULL);
   }

   return(res_scan(abuf, nmsgbuf, databuf));
}


/*
```

```
 * Skip over a compressed domain name. Return the size or -1.
 */
static                                                          180
dn_skip(comp_dn)
        char *comp_dn;
{
        register char *cp;
        register int n;

        cp = comp_dn;
        while (n = *cp++) {
                /*
                 * check for indirection                          190
                 */
                switch (n & INDIR_MASK) {
                case 0:             /* normal case, n == len */
                        cp += n;
                        continue;
                default:            /* illegal type */
                        return (-1);
                case INDIR_MASK:        /* indirection */
                        cp++;
                }                                               200
                break;
        }
        return (cp - comp_dn);
}
```

## A.1.19 hes_getpwnam.3

```
.\" Copyright 1995 by the Massachusetts Institute of Technology.  For
.\" copying and distribution information, see the file <mit-copyright.h>.
.\"
.\" Original version by Greg Hudson, MIT IS DCNS
.\"
.\" $Author: ghudson $
.\" $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/code/RCS/hes_g
.\" $Athena: hesiod.3,v 1.3 88/08/07 21:52:25 treese Locked $
.\" $Header: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/code/RCS/hes_g
.TH HESIOD 3 "28 March 1995"
.SH NAME
hes_getpwnam \- Retrieve a user password record using Hesiod resolution
.SH SYNOPSIS
.nf
.B #include <hesiod.h>
.PP
.B struct passwd *hes_getpwnam(name)
.B char *name;
.PP
.B int hes_init()
```

```
.PP
.B int hes_getpwnam_r(name, entry, buf, bufsize);
.B char *name, *buf;
.B struct passwd *entry;
.B int bufsize;
.PP
.B int hes_error()
.PP
.B cc file.c -lhesiod
.PP
.SH DESCRIPTION
.I hes_resolve
is the primary interface to the Hesiod name server.
It takes two arguments, a name to be resolved and a string, known
as a HesiodNameType.  If it succeeds, it returns a NULL-terminated vector
of strings (a la argv), one for each record containing Hesiod data.
Each such string is allocated using
.I calloc
and should be freed using
.I free
when its memory is no longer needed.  On error,
.I hes_resolve
returns NULL; the function
.I hes_error
may be called to determine the source of the error.  It will return
one of the HES_ER_* codes defined in
.I hesiod.h.  It is not necessary to call
.I hes_init
in order to use
.I hes_resolve.

.I hes_resolve_r
is a thread-safe interface to the Hesiod name server.  The caller must
pass in a return vector whose length (in number of character pointers)
is given by
.I retveclen.  hes_resolve_r
returns HES_ER_OK on success, and one of the error codes defined in
.I hesiod.h
on failure.  To use
.I hes_resolve_r,
you must call
.I hes_init()
at the beginning of your program so that it completes before any calls
to
.I hes_resolve_r()
```

```
begin.

If the environment variable
.B HES_DOMAIN
is set, this domain will override what is in /etc/athena/hesiod.conf.
.SH FILES
/usr/athena/include/hesiod.h, /etc/athena/hesiod.conf
.SH "SEE ALSO"
'Hesiod - Project Athena Technical Plan -- Name Service', named(8)
.SH AUTHOR
Steve Dyer, IBM/Project Athena
.br
Copyright 1987, 1988 by the Massachusetts Institute of Technology.
.br
Greg Hudson, MIT IS DCNS
.br
Copyright 1995 by the Massachusetts Institute of Technology.
.SH BUGS
```

### A.1.20 hesiod.3

```
.\" Copyright 1988 by the Massachusetts Institute of Technology.  For
.\" copying and distribution information, see the file <mit-copyright.h>.
.\"
.\" Original version by Steve Dyer, IBM/Project Athena.
.\"
.\" $Author: ghudson $
.\" $Source: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hesiod.3,v
.\" $Athena: hesiod.3,v 1.3 88/08/07 21:52:25 treese Locked $
.\" $Header: /afs/athena.mit.edu/user/g/h/ghudson/thesis/hesiod/RCS/hesiod.3,v
.TH HESIOD 3 "2 April 1987"
.SH NAME
hesiod \- C Language Hesiod name server Interface Library
.SH SYNOPSIS
.nf
.B #include <netdb.h>
.B #include <pwd.h>
.B #include <hesiod.h>
.PP
.B char **hes_resolve(char *\fIname\fP, char *\fItype\fP)
.B int hes_error(void)
.B char *hes_to_bind(char *\fIname\fP, char *\fItype\fP)
.B struct passwd *hes_getpwnam(char *\fIname\fP)
.B struct passwd *hes_getpwuid(int \fIuid\fP)
.B struct hes_postoffice *hes_getmailhost(char *\fIuser\fP)
```

```
.B struct servent *hes_getservbyname(char *\fIname\fP, char *\fIproto\fP)
.PP
.B int hes_init()
.B int hes_resolve_r(char *\fIname\fP, char *\fItype\fP, char **\fIretvec\fP,
int \fIretveclen\fP);
.B int hes_to_bind_r(char *\fIname\fP, char *\fItype\fP, char *\fIbuffer\fP,
int \fIbuflen\fP);
.B int hes_getpwnam_r(char *\fIname\fP, struct passwd *\fIentry\fP, char *\fIbuf\fP
int \fIbufsize\fP);
.B int hes_getpwuid_r(int \fIuid\fP, struct passwd *\fIentry\fP, char *\fIbuf\fP,
int \fIbufsize\fP);
.B int hes_getmailhost_r(char *\fIuser\fP, struct hes_postoffice *\fIret\fP,
char *\fIlinebuf\fP, int \fIbufsize\fP);
.B int hes_getservbyname_r(char *\fIname\fP, char *\fIproto\fP,
struct servent *\fIresult\fP, char *\fIbuffer\fP, int \fIbuflen\fP);
.PP
.B cc file.c -lhesiod
.PP
.SH DESCRIPTION
\fIhes_resolve\fP is the primary interface to the Hesiod name server.
It takes two arguments, a \fIname\fP to be resolved and a type of
hesiod name to resolve.  If it succeeds, it returns a NULL-terminated
vector of strings (a la argv), one for each record containing Hesiod
data.  Each such string is allocated using \fIcalloc\fP and should be
freed using \fIfree\fP when its memory is no longer needed.  On error,
\fIhes_resolve\fP returns NULL; the function \fIhes_error\fP may be
called to determine the source of the error.  It will return one of
the HES_ER_* codes defined in \fIhesiod.h\fP.

\fIhes_to_bind\fP converts a hesiod name and type into a BIND name
which may be resolved using a DNS lookup of class HESIOD.

\fIhes_getpwnam\fP, \fIhes_getpwuid\fP, and \fIhes_getservbyname\fP
look up a password or service name using Hesiod instead of the
system's local databases.  Each function works like its counterpart in
the C library.  The function \fIhes_error\fP may be called to
determine the source of errors in these functions.
\fIhes_getmailhost\fP looks up a user's mail host using Hesiod, and
returns a pointer to a struct hes_postoffice, which contains strings
describing the type, hostname, and user name of the user's mail drop:

.nf
struct hes_postoffice {
char *po_type;
char *po_host;
```

```
char *po_name;
};
.fi

The *_r versions of the Hesiod functions provide a reentrant interface
to the Hesiod library (if the DNS resolver is reentrant).  The
function \fIhes_init\fP must be called before entering any of the
reentrant Hesiod functions (this is unnecessary for the non-reentrant
versions).  Each function returns one of the HES_ER_* codes defined in
\fIhesiod.h\fP, and accepts extra parameters giving locations to place
the result of the call.  These functions will return HES_ER_RANGE if
their result buffers are not large enough to hold the result of the
lookup.  The password, mail host, and service lookup functions
require, in addition to a location for the result, a character buffer
in which to hold the data pointed to by the result structure.

If the environment variable \fBHES_DOMAIN\fP is set, this domain will
override what is in \fI/etc/athena/hesiod.conf\fP.
.SH FILES
/usr/athena/include/hesiod.h, /etc/athena/hesiod.conf
.SH "SEE ALSO"
'Hesiod - Project Athena Technical Plan -- Name Service', named(8)
.SH AUTHOR
Steve Dyer, IBM/Project Athena
.br
Copyright 1987, 1988 by the Massachusetts Institute of Technology.
.PP
Greg Hudson, MIT IS DCNS
.br
Copyright 1995 by the Massachusetts Institute of Technology.
.SH BUGS
```

## A.2 The Common Error Description Library

I am including the file `compile_et.c` because I modified it (see Chapter 4), but not the rest of the code for the `compile_et` preogram, which did not change.

### A.2.1 compile_et.c

```
/*
 *
 * Copyright 1986, 1987, 1988
 * by MIT Student Information Processing Board.
 *
 * For copyright info, see "mit-sipb-copyright.h".
```

```
 *
 */

#include <stdio.h>                                                    10
#include <sys/types.h>
#include <sys/file.h>
#include <string.h>
#include <sys/param.h>
#include "mit-sipb-copyright.h"
#include "compiler.h"

#ifndef _STDC_
#define const
#endif                                                               20

#ifndef lint
static const char copyright[] =
   "Copyright 1987,1988 by MIT Student Information Processing Board";

static const char rcsid_compile_et_c[] =
   "$Header:  /source/athena/athena.lib/et/RCS/compile_et.c,v 1.4 91/06/10 01:57:57 probe Exp $";
#endif

extern char *gensym();                                               30
extern char *current_token;
extern int table_number, current;
char buffer[BUFSIZ];
char *table_name = (char *)NULL;
FILE *hfile, *cfile;

/* C library */
extern char *malloc();
extern int errno;
                                                                     40
/* lex stuff */
extern FILE *yyin;
extern int num_lines;

char * xmalloc (size) unsigned int size; {
   char * p = malloc (size);
   if (!p) {
         perror (whoami);
         exit (1);
   }                                                                 50
   return p;
}

static int check_arg (str_list, arg) char const *const *str_list, *arg; {
   while (*str_list)
         if (!strcmp(arg, *str_list++))
            return 1;
   return 0;
}
                                                                     60
```

```
static const char *const debug_args[] = {
    "d",
    "debug",
    0,
};

static const char *const lang_args[] = {
    "lang",
    "language",
    0,
};

static const char *const language_names[] = {
    "C",
    "K&R C",
    "C++",
    0,
};

static const char * const c_src_prolog[] = {
    "static const char * const text[] = {\n",
    0,
};

static const char * const krc_src_prolog[] = {
    "#ifndef __STDC__\n",
    "#define const\n",
    "#endif\n",
    "#if !defined(__STDC__) || defined(ultrix)\n",
    "#undef sig_atomic_t\n",
    "#define sig_atomic_t int\n",
    "#endif\n",
    "\n",
    "static const char * const text[] = {\n",
    0,
};

static const char *const struct_def[] = {
    "struct error_table {\n",
    "    char const * const * msgs;\n",
    "    long base;\n",
    "    int n_msgs;\n",
    "};\n",
    "struct et_list {\n",
    "    struct et_list *next;\n",
    "    const struct error_table * table;\n",
    "};\n",
    "extern struct et_list *_et_list_ptrs[2];\n",
    "extern sig_atomic_t _et_list_index;\n",
    "\n", 0,
};

static const char warning[] =
    "/*\n * %s:\n * This file is automatically generated; please do not edit it.\n */\n";
```

```
/* pathnames */
char c_file[MAXPATHLEN];        /* output file */
char h_file[MAXPATHLEN];        /* output */

static void usage () {                                                    120
    fprintf (stderr, "%s:  usage:  %s ERROR_TABLE\n",
            whoami, whoami);
    exit (1);
}

static void dup_err (type, one, two) char const *type, *one, *two; {
    fprintf (stderr, "%s:  multiple %s specified:  '%s' and '%s'\n",
            whoami, type, one, two);
    usage ();
}                                                                         130

int main (argc, argv) int argc; char **argv; {
    char *p, *ename;
    int len;
    char const * const *cpp;
    int got_language = 0;

    /* argument parsing */
    debug = 0;
    filename = 0;                                                         140
    whoami = argv[0];
    p = strrchr (whoami, '/');
    if (p)
        whoami = p+1;
    while (argv++, --argc) {
        char *arg = *argv;
        if (arg[0] != '-') {
            if (filename)
                dup_err ("filenames", filename, arg);
            filename = arg;                                               150
        }
        else {
            arg++;
            if (check_arg (debug_args, arg))
                debug++;
            else if (check_arg (lang_args, arg)) {
                got_language++;
                arg = *++argv, argc--;
                if (!arg)
                    usage ();                                             160
                if (language)
                    dup_err ("languanges", language_names[(int)language], arg);
#define check_lang(x,v) else if (!strcasecmp(arg,x)) language = v
                check_lang ("c", lang_C);
                check_lang ("ansi_c", lang_C);
                check_lang ("ansi-c", lang_C);
                check_lang ("krc", lang_KRC);
                check_lang ("kr_c", lang_KRC);
```

```
                check_lang ("kr-c", lang_KRC);
                check_lang ("k&r-c", lang_KRC);                                    170
                check_lang ("k&r_c", lang_KRC);
                check_lang ("c++", lang_CPP);
                check_lang ("cplusplus", lang_CPP);
                check_lang ("c-plus-plus", lang_CPP);
#undef check_lang
                else {
                    fprintf (stderr, "%s:  unknown language name '%s'\n",
                            whoami, arg);
                    fprintf (stderr, "\tpick one of:  C K&R-C\n");
                    exit (1);                                                      180
                }
            }
            else {
                fprintf (stderr, "%s:  unknown control argument -'%s'\n",
                        whoami, arg);
                usage ();
            }
        }
    }
    if (!filename)                                                                 190
        usage ();
    if (!got_language)
        language = lang_KRC;
    else if (language == lang_CPP) {
        fprintf (stderr, "%s:  Sorry, C++ support is not yet finished.\n",
                 whoami);
        exit (1);
    }

    p = xmalloc (strlen (filename) + 5);                                           200
    strcpy (p, filename);
    filename = p;
    p = strrchr(filename, '/');
    if (p == (char *)NULL)
        p = filename;
    else
        p++;
    ename = p;
    len = strlen (ename);
    p += len - 3;                                                                  210
    if (strcmp (p, ".et"))
        p += 3;
    *p++ = '.';
    /* now p points to where "et" suffix should start */
    /* generate new filenames */
    strcpy (p, "c");
    strcpy (c_file, ename);
    *p = 'h';
    strcpy (h_file, ename);
    strcpy (p, "et");                                                              220

    yyin = fopen(filename, "r");
```

```
if (!yyin) {
        perror(filename);
        exit(1);
}

hfile = fopen(h_file, "w");
if (hfile == (FILE *)NULL) {
        perror(h_file);                                                    230
        exit(1);
}
fprintf (hfile, warning, h_file);

cfile = fopen(c_file, "w");
if (cfile == (FILE *)NULL) {
        perror(c_file);
        exit(1);
}
fprintf (cfile, warning, c_file);                                          240

/* prologue */
if (language == lang_C)
        cpp = c_src_prolog;
else if (language == lang_KRC)
        cpp = krc_src_prolog;
else
        abort ();
while (*cpp)
        fputs (*cpp++, cfile);                                             250

/* parse it */
yyparse();
fclose(yyin);                       /* bye bye input file */

fputs ("      0\n};\n\n", cfile);
for (cpp = struct_def; *cpp; cpp++)
        fputs (*cpp, cfile);
fprintf(cfile,
          "static const struct error_table et = { text, %ldL, %d };\n\n",  260
          table_number, current);
fputs("static struct et_list link = { 0, 0 };\n\n", cfile);
fprintf(cfile, "void initialize_%s_error_table() {\n", table_name);
fputs("     if (!link.table) {\n", cfile);
fputs("          link.next = _et_list_ptrs[_et_list_index];\n", cfile);
fputs("          link.table = &et;\n", cfile);
fputs("          _et_list_ptrs[!_et_list_index] = &link;\n", cfile);
fputs("          _et_list_index = !_et_list_index;\n", cfile);
fputs("     }\n", cfile);
fputs("}\n", cfile);                                                       270
fclose(cfile);

fprintf (hfile, "extern void initialize_%s_error_table ();\n",
          table_name);
fprintf (hfile, "#define ERROR_TABLE_BASE_%s (%ldL)\n",
          table_name, table_number);
```

```
    /* compatibility... */
    fprintf (hfile, "\n/* for compatibility with older versions...  */\n");
    fprintf (hfile, "#define init_%s_err_tbl initialize_%s_error_table\n",
            table_name, table_name);
    fprintf (hfile, "#define %s_err_base ERROR_TABLE_BASE_%s\n", table_name,
            table_name);
    fclose(hfile);                  /* bye bye include file */

    return 0;
}

int yyerror(s) char *s; {
    fputs(s, stderr);
    fprintf(stderr, "\nLine number %d; last token was '%s'\n",
            num_lines+1, current_token);
}

#if 0 /* previously "i386", which is wrong; add OS defines as appropriate. */
/* Need strcasecmp for this machine */
/*
 * Copyright (c) 1987 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

#if defined(LIBC_SCCS) && !defined(lint)
static char sccsid[] = "@(#)strcasecmp.c        1.3 (Berkeley) 8/3/87";
#endif LIBC_SCCS and not lint

/*
 * This array is designed for mapping upper and lower case letter
 * together for a case independent comparison.  The mappings are
 * based upon ascii character sequences.
 */
static char charmap[] = {
        '\000', '\001', '\002', '\003', '\004', '\005', '\006', '\007',
        '\010', '\011', '\012', '\013', '\014', '\015', '\016', '\017',
        '\020', '\021', '\022', '\023', '\024', '\025', '\026', '\027',
        '\030', '\031', '\032', '\033', '\034', '\035', '\036', '\037',
        '\040', '\041', '\042', '\043', '\044', '\045', '\046', '\047',
        '\050', '\051', '\052', '\053', '\054', '\055', '\056', '\057',
        '\060', '\061', '\062', '\063', '\064', '\065', '\066', '\067',
        '\070', '\071', '\072', '\073', '\074', '\075', '\076', '\077',
        '\100', '\141', '\142', '\143', '\144', '\145', '\146', '\147',
        '\150', '\151', '\152', '\153', '\154', '\155', '\156', '\157',
        '\160', '\161', '\162', '\163', '\164', '\165', '\166', '\167',
        '\170', '\171', '\172', '\133', '\134', '\135', '\136', '\137',
        '\140', '\141', '\142', '\143', '\144', '\145', '\146', '\147',
        '\150', '\151', '\152', '\153', '\154', '\155', '\156', '\157',
        '\160', '\161', '\162', '\163', '\164', '\165', '\166', '\167',
        '\170', '\171', '\172', '\173', '\174', '\175', '\176', '\177',
        '\200', '\201', '\202', '\203', '\204', '\205', '\206', '\207',
        '\210', '\211', '\212', '\213', '\214', '\215', '\216', '\217',
        '\220', '\221', '\222', '\223', '\224', '\225', '\226', '\227',
```

```
        '\230', '\231', '\232', '\233', '\234', '\235', '\236', '\237',
        '\240', '\241', '\242', '\243', '\244', '\245', '\246', '\247',
        '\250', '\251', '\252', '\253', '\254', '\255', '\256', '\257',
        '\260', '\261', '\262', '\263', '\264', '\265', '\266', '\267',
        '\270', '\271', '\272', '\273', '\274', '\275', '\276', '\277',
        '\300', '\341', '\342', '\343', '\344', '\345', '\346', '\347',
        '\350', '\351', '\352', '\353', '\354', '\355', '\356', '\357',
        '\360', '\361', '\362', '\363', '\364', '\365', '\366', '\367',
        '\370', '\371', '\372', '\333', '\334', '\335', '\336', '\337',
        '\340', '\341', '\342', '\343', '\344', '\345', '\346', '\347',
        '\350', '\351', '\352', '\353', '\354', '\355', '\356', '\357',
        '\360', '\361', '\362', '\363', '\364', '\365', '\366', '\367',
        '\370', '\371', '\372', '\373', '\374', '\375', '\376', '\377',
};

strcasecmp(s1, s2)
        register char *s1, *s2;
{
        register char *cm = charmap;

        while (cm[*s1] == cm[*s2++])
                if (*s1++ == '\0')
                        return(0);
        return(cm[*s1] - cm[*--s2]);
}

#endif
```

## A.2.2 error_message.c

```
/*
 * $Header: error_message.c,v 1.2 89/01/25 09:08:57 shanzer Exp $
 * $Source: /paris/source/4.3/athena.lib/et.new/RCS/error_message.c,v $
 * $Locker:  $
 *
 * Copyright 1987 by the Student Information Processing Board
 * of the Massachusetts Institute of Technology
 *
 * For copyright info, see "mit-sipb-copyright.h".
 */

#include <stdio.h>
#include <signal.h>
#include "error_table.h"
#include "mit-sipb-copyright.h"
#include "internal.h"
#include "com_err.h"

static const char rcsid[] =
   "$Header:  error_message.c,v 1.2 89/01/25 09:08:57 shanzer Exp $";
static const char copyright[] =
   "Copyright 1986, 1987, 1988 by the Student Information Processing Board\nand the department of
```

```
#if !defined(_STDC_) || d          fined(ultrix)
#undef sig_atomic_t
#define sig_atomic_t int
#endif

/* To allow atomic updates        / to the list, we have two potential pointers to the
 * head of the list and an in     dex telling us which one to use.  To atomically
 * update the list, modify _e     t_list_ptrs[!_et_list_index] and then invert
 * _et_list_index. */
struct et_list * _et_list_ptrs     [2] = { NULL, NULL };
sig_atomic_t _et_list_index =      0;

const char * error_message         (code)
long     code;
{
   static char buf[COM_E           RR_BUF_LEN];

   return(error_message_r(         code, buf));
}

const char * error_message         _r (code, buf)
long     code;
char     *buf;
{
   int offset;
   struct et_list *et;
   int table_num;
   int started = 0;
   char *cp, namebuf[6];

   offset = code & ((1<<E          RRCODE_RANGE)-1);
   table_num = code - off          set;
   if (!table_num)
         return strerror(o         ffset);
   for (et = _et_list_ptrs[_e      t_list_index]; et; et = et->next) {
         if (et->table->t          base == table_num) {
            /* This is the          right table */
            if (et->table-          ->n_msgs <= offset)
                  goto oo           ps;
            return(et->t            able->msgs[offset]);
         }
   }
oops:
   strcpy (buf, "Unknown c         code ");
   if (table_num) {
         strcat (buf, error          table_name_r (table_num, namebuf));
         strcat (buf, " ");
   }
   for (cp = buf; *cp; cp++        ++)
         ;
   if (offset >= 100) {
         *cp++ = '0' + o           offset / 100;
         offset %= 100;
         started++;
```

```
    }
    if (started || offset >= 10) {
            *cp++ = '0' + offset / 10;
            offset %= 10;
    }
    *cp++ = '0' + offset;
    *cp = '\0';
    return(buf);
}
```

## A.2.3 et_name.c

```
/*
 * Copyright 1987 by MIT Student Information Processing Board
 *
 * For copyright info, see mit-sipb-copyright.h.
 */

#include "error_table.h"
#include "mit-sipb-copyright.h"
#include "internal.h"

#ifndef lint
static const char copyright[] =
    "Copyright 1987,1988 by Student Information Processing Board, Massachusetts Institute of Techno
static const char rcsid_et_name_c[] =
    "$Header:  et_name.c,v 1.7 89/01/01 06:14:56 raeburn Exp $";
#endif

static const char char_set[] =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_";

const char * error_table_name_r(num, buf)
    int num;
    char *buf;
{
    int ch;
    int i;
    char *p;

    /* num = aa aaa abb bbb bcc ccc cdd ddd d?? ??? ??? */
    p = buf;
    num >>= ERRCODE_RANGE;
    /* num = ?? ??? ??? aaa aaa bbb bbb ccc ccc ddd ddd */
    num &= 077777777;
    /* num = 00 000 000 aaa aaa bbb bbb ccc ccc ddd ddd */
    for (i = 4; i >= 0; i--) {
        ch = (num >> BITS_PER_CHAR * i) & ((1 << BITS_PER_CHAR) - 1);
        if (ch != 0)
            *p++ = char_set[ch-1];
    }
    *p = '\0';
    return(buf);
```

```
}

const char * error_table_name(num)
    int num;
{
    static char buf[6];

    return(error_table_name_r(num, buf));
}
```

## A.2.4 init_et.c

```
/*
 * $Header: init_et.c,v 1.5 88/10/27 08:34:54 raeburn Exp $
 * $Source: /mit/raeburn/Work/et/src/RCS/init_et.c,v $
 * $Locker:  $
 *
 * Copyright 1986, 1987, 1988 by MIT Information Systems and
 *        the MIT Student Information Processing Board.
 *
 * For copyright info, see mit-sipb-copyright.h.
 */

#include <stdio.h>
#include "error_table.h"
#include "mit-sipb-copyright.h"

#ifndef _STDC_
#define const
#endif

#ifndef lint
static const char rcsid_init_et_c[] =
    "$Header:  init_et.c,v 1.5 88/10/27 08:34:54 raeburn Exp $";
#endif

extern char *malloc(), *realloc();

struct foobar {
    struct et_list etl;
    struct error_table et;
};

extern struct et_list * _et_list;

int init_error_table(msgs, base, count)
    const char * const * msgs;
    int base;
    int count;
{
    struct foobar * new_et;
```

```
    if (!base || !count || !msgs)
        return 0;

    new_et = (struct foobar *) malloc(sizeof(struct foobar));
    if (!new_et)
        return errno;     /* oops */
    new_et->etl.table = &new_et->et;
    new_et->et.msgs = msgs;
    new_et->et.base = base;
    new_et->et.n_msgs= count;                                    50

    new_et->etl.next = _et_list;
    _et_list = &new_et->etl;
    return 0;
}
```

## A.2.5 com_err.c

```
/*
 * Copyright 1987, 1988 by MIT Student Information Processing Board.
 *
 * For copyright info, see mit-sipb-copyright.h.
 */

#include <stdio.h>
#include "mit-sipb-copyright.h"

#if !defined(_STDC_) && defined(_vax_)                           10
#define VARARGS 1
#endif

#ifdef _vax_
/* We don't have the v*printf routines... */
#define vfprintf(stream,fmt,args) _doprnt(fmt,args,stream)
#endif

#if !defined(VARARGS)
#       include <stdarg.h>                                       20
#else /* varargs: not STDC or no <stdarg> */
        /* Non-ANSI, always take <varargs.h> path. */
#       undef VARARGS
#       define VARARGS 1
#       include <varargs.h>
#endif /* varargs */

#include "error_table.h"
#include "internal.h"

                                                                 30
/*
 * Protect us from header version (externally visible) of com_err, so
 * we can survive in a <varargs.h> environment.  I think.
 */
#define com_err com_err_external
```

```
#include "com_err.h"
#undef com_err

#if ! lint
static const char rcsid[] =                                                    40
   "$Header:  /afs/rel-eng.athena.mit.edu/project/release/source/athena/athena.lib/et/RCS/com_e
#endif /* ! lint */

static void
#ifdef _STDC_
   default_com_err_proc (const char *whoami, long code, const char *fmt, va_list args)
#else
   default_com_err_proc (whoami, code, fmt, args)
   const char *whoami;
   long code;                                                                  50
   const char *fmt;
   va_list args;
#endif
{
   char buf[25];

   if (whoami) {
         fputs(whoami, stderr);
         fputs(":  ", stderr);
   }                                                                           60
   if (code) {
         fputs(error_message_r(code, buf), stderr);
         fputs(" ", stderr);
   }
   if (fmt) {
      vfprintf (stderr, fmt, args);
   }
   /* should do \r only on a tty in raw mode, but it won't hurt */
   putc('\r', stderr);
   putc('\n', stderr);                                                         70
   fflush(stderr);
}

error_handler_t com_err_hook = default_com_err_proc;

void com_err_va (whoami, code, fmt, args)
   const char *whoami;
   long code;
   const char *fmt;
   va_list args;                                                               80
{
   (*com_err_hook) (whoami, code, fmt, args);
}

#if ! VARARGS
void com_err (const char *whoami,
              long code,
              const char *fmt, ...)
{
```

```
#else                                                          90
void com_err (va_alist)
    va_dcl
{
    const char *whoami, *fmt;
    long code;
#endif
    va_list pvar;

#if VARARGS
    va_start (pvar);                                           100
    whoami = va_arg (pvar, const char *);
    code = va_arg (pvar, long);
    fmt = va_arg (pvar, const char *);
#else
    va_start(pvar, fmt);
#endif
    com_err_va (whoami, code, fmt, pvar);
    va_end(pvar);
}
                                                               110
error_handler_t set_com_err_hook (new_proc)
    error_handler_t new_proc;
{
    error_handler_t x = com_err_hook;

    if (new_proc)
        com_err_hook = new_proc;
    else
        com_err_hook = default_com_err_proc;
                                                               120
    return x;
}

error_handler_t reset_com_err_hook () {
    error_handler_t x = com_err_hook;
    com_err_hook = default_com_err_proc;
    return x;
}
```

## A.2.6 com_err.3

```
.\" Copyright (c) 1988 Massachusetts Institute of Technology,
.\" Student Information Processing Board.  All rights reserved.
.\"
.\" $Header$
.\"
.TH COM_ERR 3 "22 Nov 1988" SIPB
.SH NAME
com_err \- common error display routine
.SH SYNOPSIS
.nf
```

```
 #include <com_err.h>
.PP
void initialize_XXXX_error_table(void);
.PP
void com_err(const char *\fIwhoami\fP, long \fIcode\fP,
const char *\fIformat\fP, ...);
.PP
const char *error_message(long \fIcode\fP);
.PP
const char *error_message_r(long \fIcode\fP, char *\fIbuf\fP);
.PP
typedef void(*error_handler_t)(const char *\fIwhoami\fP,
long \fIcode\fP, const char *\fIformat\fP, va_list \fIargs\fP);
.PP
error_handler_t set_com_err_hook(error_handler_t \fIproc\fP);
.PP
error_handler_t reset_com_err_hook(void);
.fi
.SH DESCRIPTION
\fICom_err\fP displays an error message on the standard error stream
\fIstderr\fP (see
.IR stdio (3S))
composed of the \fIwhoami\fP string, which should specify the program
name or some subportion of a program, followed by an error message
generated from the \fIcode\fP value (derived from
.IR compile_et (1)),
and a string produced using the \fIformat\fP string and any following
arguments, in the same style as
.IR fprintf (3).

The behavior of \fIcom_err\fP can be modified using
\fIset_com_err_hook\fP; this defines a procedure which is called with
the arguments passed to \fIcom_err\fP, instead of the default internal
procedure which sends the formatted text to error output.  Thus the
error messages from a program can all easily be diverted to another
form of diagnostic logging, such as
.IR syslog (3).
\fIReset_com_err_hook\fP may be used to restore the behavior of
\fIcom_err\fP to its default form.  Both procedures return the
previous ``hook'' value.  These ``hook'' procedures must have the
declaration given for \fIproc\fP above in the synopsis.

The \fIinitialize_XXXX_error_table\fP routine is generated
mechanically by
.IR compile_et (1)
```

```
from a source file containing names and associated strings.  Each
table has a name of up to four characters, which is used in place of
the \fBXXXX\fP in the name of the routine.  These routines should be
called before any of the corresponding error codes are used, so that
the \fIcom_err\fP library will recognize error codes from these tables
when they are used.

The functions \fIerror_message\fP and \fIerror_message_r\fP return the
error message for an error code.  \fIerror_message\fP may return its
error message inside a static buffer, and therefore is not reentrant;
\fIerror_message_r\fP will in these cases use the storage provided in
\fIbuf\fP, which should be at least \fBCOM_ERR_BUF_LEN\fP bytes long.

The \fBcom_err.h\fP header file should be included in any source file
that uses routines from the \fIcom_err\fP library; executable files
must be linked using \fI``-lcom_err''\fP in order to cause the
\fIcom_err\fP library to be included.

Multithreaded programs must perform all of the
\fIinitialize_XXXX_error_table\fP calls serially, but may perform them
in parallel with \fIcom_err\fP, \fIerror_message\fP, and
\fIerror_mesesage_r\fP calls as long as the call to
\fIinitialize_XXX_error_table\fP for a given table has completed
before any corresponding error codes are used.  Calls to
\fIset_com_err_hook\fP and \fIreset_com_err_hook\fP must also be
performed serially or they may return the wrong old hook value.

.\" .IR for manual entries
.\" .PP for paragraph breaks

.SH "SEE ALSO"
compile_et (1), syslog (3).

Ken Raeburn, "A Common Error Description Library for UNIX".
```

## A.3 Attach

I've omitted the manual pages, since I didn't modify any of them. I have also omitted the three RVD files, since they don't result in any code being compiled in.

### A.3.1 Imakefile

```
#ifdef SOLARIS
OSDEF= -D_REENTRANT -DINTERNAL_RESOLVER -DSOLARIS -DPOSIX -Isolaris
```

```
LDLIBS= -lsocket -lnsl -lthread
#else
CC=pgcc
LD=pgcc
#endif

CDEBUG=-g
DEFINES=-I/mit/gnu/include $(OSDEF)
XOBJS=/mit/gnu/arch/${ATHENA_SYS}/lib/regex.o

#ifdef _AUX_SOURCE
XSRCS = emul_re.c
XOBJS = emul_re.o
XLIBS = -lPW -lc
#endif

#if !defined(ultrix)
RPCLIB=-lrpcsvc
#endif

LDDEFS = -L../hesiod -L../et
LIBS = -lhesiod -lzephyr -lkrb -ldes $(RPCLIB) -lcom_err $(XLIBS)

SRCS = main.c util.c attachtab.c attach.c detach.c mul.c nfs.c rvd.c\
afs.c ufs.c rpc.c mount.c unmount.c zephyr.c getrealm.c\
rvdutil.c pathcan.c config.c $(XSRCS)
OBJS =  main.o util.o attachtab.o attach.o detach.o mul.o nfs.o rvd.o\
afs.o ufs.o rpc.o mount.o unmount.o zephyr.o getrealm.o\
rvdutil.o pathcan.o config.o $(XOBJS)


build_program(attach,$(OBJS),,$(LIBS))
install_program(attach,-m 4755 -o root,$(ATHRBINDIR))

install_man(attach.1,attach.1)
install_man(detach.1,detach.1)
install_man(fsid.1,fsid.1)
install_man(attach.conf.5,attach.conf.5)
install_man(attachtab.5,attachtab.5)
install_man(zinit.8,zinit.8)

install_man_links(fsid.1,nfsid.1)

install::
$(RM) $(DESTDIR)$(ATHRBINDIR)/detach
```

```
$(RM) $(DESTDIR)$(ATHRBINDIR)/fsid
$(RM) $(DESTDIR)$(ATHRBINDIR)/nfsid
$(RM) $(DESTDIR)$(ATHRBINDIR)/zinit
$(RM) $(DESTDIR)$(ATHRETCDIR)/zinit
ln -s attach $(DESTDIR)$(ATHRBINDIR)/detach
ln -s attach $(DESTDIR)$(ATHRBINDIR)/fsid
ln -s attach $(DESTDIR)$(ATHRBINDIR)/nfsid
ln -s attach $(DESTDIR)$(ATHRBINDIR)/zinit
ln -s $(ATHRBINDIR)/attach $(DESTDIR)$(ATHRETCDIR)/zinit

protos.h: protos.h.new
@-cmp -s $@ protos.h.new || \
(echo cp protos.h.new $@; cp protos.h.new $@)

protos.h.new: $(SRCS)
mkptypes -p __P -z $(SRCS) > $@

$(OBJS): protos.h
```

## A.3.2 attach.h

```
/*
 * $Id: attach.h,v 1.17 93/06/30 16:52:17 vrt Exp $
 *
 * Copyright (c) 1988,1991 by the Massachusetts Institute of Technology.
 *
 * For redistribution rights, see "mit-copyright.h"
 */

#include "config.h"

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <errno.h>
#ifdef _AIX
#undef geterrno
#endif
#include <netdb.h>
#include <strings.h>

#include <sys/types.h>
#include <sys/file.h>
#include <sys/param.h>
#include <sys/socket.h>
#include <sys/time.h>
```

```
#include <netinet/in.h>

#ifdef NFS                                                          30
#include <rpc/rpc.h>
#ifndef i386
#include <nfs/nfs.h>
#else
#include <rpc/nfs.h>
#endif
#ifdef NeXT
#include <nfs/nfs_mount.h>              /* Newer versions of NFS (?) */
#endif /* NeXT */
#include <rpcsvc/mount.h>                                           40
#ifdef _AUX_SOURCE
#include <nfs/mount.h>
#endif
#ifdef SOLARIS
#include <rpc/clnt_soc.h>
#include <nfs/mount.h>
#include <sys/fs/ufs_mount.h>
#endif
#endif /* NFS */
                                                                    50
#ifdef ultrix
#include <ufs/ufs_mount.h>
#ifdef NFS
#include <nfs/nfs_gfs.h>
#endif /* NFS */
#define KERNEL
/* AACK!  this @#OU@#)($(#) file defines and initializes an array which
   loses on multiple includes, but it's surrounded by #ifdef KERNEL. */
#include <sys/fs_types.h>
#undef KERNEL                                                       60
#endif /* ultrix */

#include <sys/mount.h>
#ifdef _AIX
#include <sys/vmount.h>
#define M_RDONLY      MNT_READONLY
#endif

#if defined(_AUX_SOURCE) || defined(NeXT) || defined(_AIX)
#define vfork   fork                                                70
#endif

/*
 * If MOUNT_CMD or UMOUNT_CMD are defined, it will run the program
 * specified rather than trying to use in-line code.
 */
#if defined(_IBMR2)
#define MOUNT_CMD   "/etc/mount"
#endif
#if defined(SOLARIS)                                                80
#define MOUNT_CMD "/etc/fs/nfs/mount"
```

```
#define UMOUNT_CMD "/usr/sbin/umount"
#endif

#define MAXOWNERS 64
#define MAXHOSTS 64

#ifndef _P
# ifdef _STDC_
#  define _P(x) x                                                    90
# else
#  define _P(x)
# endif
#endif

/*
 * We don't really want to deal with malloc'ing and free'ing stuff
 * in this structure...
 */
                                                                     100
struct _attachtab {
        struct _attachtab       *next, *prev;
        pthread_mutex_t         lock;

        char                    version[3];
        char                    explicit;
        char                    status;
        char                    mode;
        struct _fstypes         *fs;
        struct                  in_addr hostaddr[MAXHOSTS];          110
        int                     rmdir;
        int                     drivenum;
        int                     flags;
        int                     nowners;
        uid_t                   owners[MAXOWNERS];
        char                    hesiodname[BUFSIZ];
        char                    host[BUFSIZ];
        char                    hostdir[MAXPATHLEN];
        char                    mntpt[MAXPATHLEN];
};                                                                   120

/* State for storing an in-memory list of attachtab entries. */
struct attach_list {
        struct _attachtab       *attachtab_first;
        struct _attachtab       *attachtab_last;
        int                     attach_lock_fd;
        int                     attach_lock_count;
};
#define ATTACH_LIST_INITIALIZER { NULL, NULL, -1, 0 }
                                                                     130
struct fslock {
   int fd;
   char filename[BUFSIZ];
};
```

```
/*
 * Attach flags defines
 *
 * FLAG_NOSETUID --- this filesystem was mounted nosetuid (no meaning
 *       for afs filesystems)
 * FLAG_LOCKED --- this filesystem is passed over by detach -a, and
 *       you must be the owner of the filesystem to detach it.
 * FLAG_ANYONE --- anyone can detach this filesystem  (not yet implemented)
 * FLAG_PERMANENT --- when this filesytem is detached, don't do
 *       actually unmount it; just deauthenticate, if necessary.  attach
 *       sets this flag if it finds the filesystem already mounted but
 *       not in attachtab.
 */
#define FLAG_NOSETUID 1
#define FLAG_LOCKED 2
#define FLAG_ANYONE 4
#define FLAG_PERMANENT 8

#define ATTACH_VERSION "A1"

#define ATTACHTABMODE 644

#define STATUS_ATTACHED       '+'
#define STATUS_ATTACHING   '*'
#define STATUS_DETACHING   '-'

#define TYPE_NFS      001
#define TYPE_RVD      002
#define TYPE_UFS      004
#define TYPE_ERR      010
#define TYPE_AFS      020
#define TYPE_MUL      040
#define ALL_TYPES     067

/*
 * Attach configuration defines
 */
#define MAXFILTAB    100
#define MAXTRUIDTAB 100

/* Values set during initialization phase and never changed. */
struct config {
   /* Values set in config.c while reading attachtab.conf. */
   int owner_check;
   int owner_list;
   int keep_mount;
   int exp_mntpt;
   int exp_allow;
   int nfs_root_hack;
   char *nfs_mount_dir;
   char *attachtab_fn;
   char *mtab_fn;
#ifdef AFS
   char *aklog_fn;
```

```
    char *afs_mount_dir;
#endif
    char *fsck_fn;

    /* Set in config.c; ONLY used to initialize flags. */
    int verbose;
    int debug_flag;

    /* Values set at the begining of main(). */
    char *progname;
    int real_uid;
    int effective_uid;
    char *abort_msg;

    /* Values set at the beginning of attachcmd(). */
    int default_suid;
};

/* Flags varying per attach. */
struct flags {
    /* Values set up by command line and never changed. */
    int verbose;
    int debug_flag;
    int map_anyway;
    int do_nfsid;
    int print_path;
    int explicit;
    int override;
    int clean_detach;
    int force;
    int lock_filesystem;
    int lookup;
#ifdef ZEPHYR
    int use_zephyr;
#endif
    int override_mode;
    char *mount_options;
    char *filsys_type;
    char *mntpt;
    int override_suid;
    char *spoofhost;
    int owner_uid;
    int skip_fsck;

    /* Communication between main functions and threads. */
    char outbuf[200];
    char *fsname;
    char *output;
    int status;
};

/*
 * Mount options
 */
```

```
#ifndef M_RDONLY
#define M_RDONLY    0x01            /* mount fs read-only */
#endif
#ifndef M_NOSUID
#define M_NOSUID    0x02            /* mount fs without setuid perms */
#endif

/*
 * Mount option table
 */
struct mntopts {
        int     type;               /* File system type */
        int     flags;              /* Mount flags */
#ifdef NFS
        int     nfs_port;           /* Valid only for NFS, port for rpc.mountd */
#endif
        union tsa {
#ifdef UFS
                struct ufs_args   ufs;
#endif
#ifdef NFS
                struct nfs_args nfs;
#endif
        } tsa;
};

/*
 * Type table
 */

struct _fstypes {
   char         *name;
   int          type;
   int          mount_type;
   int          flags;
   char         *good_flags;
   int          (*attach) _P((struct _attachtab *at, struct mntopts *mopt,
                              int errorout, struct flags *flags));
   int          (*detach) _P((struct _attachtab *at, struct attach_list *list,
                              struct flags *flags));
   int          (*explicit) _P((char *name, char **hesptr, char *hesline,
                                struct flags *flags));
};

/*
 * Flags for _fstypes.flags
 */
#define AT_FS_MNTPT         1
#define AT_FS_REMOTE        2
#define AT_FS_PARENTMNTPT   4
#define AT_FS_MNTPT_CANON   8

/*
 * Command option lists
```

```
 */

struct command_list {                                          300
    char            *large;
    char            *small;
};

/*
 * RVD defines
 */

#define RVD_ATTACH_TIMEOUT 30
                                                               310
/*
 * RPC caching
 */

#define RPC_MAXCACHE 10

struct cache_ent {
    struct          in_addr addr;
    CLIENT          *handle;
    struct          sockaddr_in sin;                           320
    int             fd;
    int             error;
};

/*
 * Calls to RPC.MOUNTD
 */

#ifndef MOUNTPROC_KUIDMAP
#define MOUNTPROC_KUIDMAP 7                                    330
#define MOUNTPROC_KUIDUMAP 8
#define MOUNTPROC_KUIDPURGE 9
#define MOUNTPROC_KUIDUPURGE 10
#endif

/*
 * Command names
 */

#define ATTACH_CMD  "attach"                                   340
#define DETACH_CMD  "detach"
#define NFSID_CMD   "nfsid"
#define FSID_CMD    "fsid"
#ifdef ZEPHYR
#define ZINIT_CMD   "zinit"
#endif /* ZEPHYR */

/*
 * Generic defines
 */                                                            350
```

```
#define SUCCESS 0
#define FAILURE 1

/*
 * Error status defininitions
 */

#define ERR_NONE      0        /* No error */
#define ERR_BADARGS 1          /* Bad arguments */                    360
#define ERR_SOMETHING 2        /* Something wrong - > 1 args */
#define ERR_FATAL     3        /* Internal failure */
#define ERR_INTERRUPT 4        /* Program externally aborted */
#define ERR_BADCONF 5          /* Bad configuration file */
#define ERR_BADFSDSC 6         /* Bad filesystem description */
#define ERR_BADFSFLAG 7        /* Bad filsys flag */

#define ERR_KERBEROS 10        /* Kerberos failure */
#define ERR_HOST      11       /* General host communication failure */
#define ERR_AUTHFAIL 12        /* Authentication failure */           370
#define ERR_NOPORTS 13         /* Out of reserved ports */

#define ERR_NFSIDNOTATTACHED 20 /* Filesystem with -f not attached */
#define ERR_NFSIDBADHOST 21 /* Can't resolve hostname */
#define ERR_NFSIDPERM 22    /* unauthorized nfsid -p */

#define ERR_ATTACHBADFILSYS 20 /* Bad filesystem name */
#define ERR_ATTACHINUSE 21 /* Filesystem in use by another proc */
#define ERR_ATTACHNEEDPW 22 /* RVD spinup needs a password */
#define ERR_ATTACHFSCK 23  /* FSCK returned error on RVD */           380
#define ERR_ATTACHNOTALLOWED 24 /* User not allowed to do operation */
#define ERR_ATTACHBADMNTPT 25 /* User not allowed to mount a */
                              /* filesystem here */
#define ERR_ATTACHNOFILSYS 26 /* The remote filesystem doesn't exist */
#define ERR_ATTACHDIRINUSE 27 /* Some other filesystem is using the */
                      /* mountpoint directory */

#define ERR_DETACHNOTATTACHED 20 /* Filesystem not attached */
#define ERR_DETACHINUSE 21 /* Filesystem in use by another proc */
#define ERR_DETACHNOTALLOWED 22 /* User not allowed to do operations */ 390

#define ERR_ZINITZLOSING    20       /* Random zephyr lossage */

/*
 * Zephyr definitions
 */

#ifdef ZEPHYR
#define ZEPHYR_CLASS "filsrv"
#define ZEPHYR_MAXSUBS 100 /* 50 filesystems... */                    400
#define ZEPHYR_TIMEOUT  60 /* 1 minute timeout */
#endif /* ZEPHYR */

#ifndef _P
#ifdef _STDC_
```

```
#define _P(x) x
#else
#define _P(x) ()
#endif
#endif

extern struct config config;
extern struct _fstypes fstypes[];

#ifdef NFS
#include <krb.h>
#endif

#include "protos.h"
```

## A.3.3 config.h

```
/*
 * Contains the local configuration information for attach/detach/nfsid
 *      $Id: config.h,v 1.15 93/05/05 17:04:52 vrt Exp $
 *
 * Configuration defines
 *
 * Warning:  attach may not compile if NFS is not defined... given
 *      that attach was originally designed just for NFS filesystems,
 *      this isn't so surprising.  Sigh.  This is true, to a lesser
 *      extent, for KERBEROS.
 *
 * NEED_STRTOK means that we're on a system that does not have
 * strtok() support in libc (usually pre BSD 4.3 systems).
 *
 * OLD_KERBEROS means we're compiling with the old, buggy kerberos
 * library.  This is necessary because of release skew.  Note that the
 * same gratuitous name changes took place between the new and old
 * kerberos libraries.
 */

#define NFS
#define AFS

#if (defined(vax) && !defined(ultrix)) || defined(ibm032) || defined(i386)
/* #define       RVD */
#endif
#if defined(sun)
#define NFSCLIENT
#endif
#if !defined(_AIX)
#define UFS
#endif

#define ZEPHYR
#define HESIOD
```

```
#define KERBEROS

#define ATHENA_COMPAT73     /* 7.3 fsid compatibility */

/*                                                                    40
 * Other external filenames
 */
#define ATTACHCONFFILE "/etc/athena/attach.conf"
#define ATTACHTAB    "/usr/tmp/attachtab"
#define FSCK_FULLNAME "/etc/fsck"
#define FSCK_SHORTNAME "fsck"
#define AKLOG_FULLNAME "/bin/athena/aklog"
#define AKLOG_SHORTNAME "aklog"
#define RVDGETM_FULLNAME "/etc/athena/rvdgetm"
#define RVDGETM_SHORTNAME "rvdgetm"                                   50

#if defined(_AIX) && defined(i386)
#define MTAB            "/local/mtab"
#else
#if defined(SOLARIS)
#define MTAB            "/etc/mnttab"
#else
#define MTAB            "/etc/mtab"
#endif
#endif                                                                60

/*
 * Kerberos instance
 */
#if defined(KERBEROS) && defined(NFS)
#define KERB_NFSID_INST "rvdsrv"
#endif

/*
 * Default mount directory for afs                                    70
 */
#ifdef AFS
#define AFS_MOUNT_DIR "/mit"
#endif

/*
 * This is the type of function required by signal.  Since it changes
 * from system to system, it is declared here.
 */
#if defined(POSIX) && !defined(vax)                                   80
typedef void     sig_catch;
#else
typedef int      sig_catch;
#if !defined(sun)
char *malloc();
#endif
#endif

#if defined(ultrix) && !defined(vax)
```

```
/* Ultrix vax compiler complains about void * casting */
#include <malloc.h>
#endif

/*
 * This is a set of horrible hacks to make attach support Ultrix as
 * easily as possible.  Praise (and Blame) belongs to John Kohl.
 */
#ifdef ultrix
/* need to re-name some structures for Ultrix */
#define ufs_args        ufs_specific
#define nfs_args        nfs_gfs_mount

/* define a struct mntent for convenience of mount.c & unmount.c.  Used
   mainly as a convenient place to store pointers to stuff needed
   for the Ultrix mount() and umount() syscalls. */

struct mntent {
    char *mnt_fsname;
    char *mnt_dir;
    int mnt_type;
    char *mnt_opts;
    int mnt_freq;
    int mnt_passno;
};

/* hacks for filesystem type */
#define MOUNT_NFS   GT_NFS
#define MOUNT_UFS   GT_ULTRIX

/* hacks for M_ names */
#define M_RDONLY    M_RONLY

#define PGUNITS         1024    /* to convert MINPGTHRESH to K */
#define DEFPGTHRESH 64      /* default page threshhold */

#endif /* ultrix compat stuff */

/* These are not defined or recognized by the system, but they are useful
   to allow common data structures with systems that do have these defines */
#if defined(_AIX) || defined(sun)
#define MOUNT_UFS   1
#define MOUNT_NFS   2
#endif
```

## A.3.4 solaris/pthread.h

```
/* System includes for Solaris threads. */
#include <thread.h>
#include <synch.h>

/* Constants and types. */
#define _POSIX_THREADS
```

```
#define _POSIX_THREAD_ATTR_STACKSIZE
#define _POSIX_THREAD_ATTR_STACKADDR
#define PTHREAD_STACK_MIN               thr_min_stack()
#define PTHREAD_CREATE_DETACHED         THR_DETACHED
#define PTHREAD_CREATE_JOINABLE         0
#define PTHREAD_MUTEX_INITIALIZER       SHAREDMUTEX
#define PTHREAD_COND_INITIALIZER        SHAREDCV
#define PTHREAD_ONCE_INIT               { PTHREAD_MUTEX_INITIALIZER, 0 }
typedef thread_t pthread_t;
typedef mutex_t pthread_mutex_t;
typedef cond_t pthread_cond_t;
typedef thread_key_t pthread_key_t;
typedef struct { int size; void *base; int flags; } pthread_attr_t;
typedef struct { pthread_mutex_t lock; int flag; } pthread_once_t;

/* Translations for attribute initialization and thread creation. */
#define pthread_attr_init(a)            ((a)->base = 0, (a)->size = 0, \
                                            (a)->flags = THR_BOUND)
#define pthread_attr_destroy(a)
#define pthread_attr_setstacksize(a, b)         ((a)->size = (b))
#define pthread_attr_getstacksize(a)    ((a)->size)
#define pthread_attr_setstackaddr(a, b) ((a)->base = (b))
#define pthread_attr_getstackaddr(a)    ((a)->base)
#define pthread_attr_setdetachstate(a, b) ((a)->flags = (b))
#define pthread_attr_getdetachstate(a)  ((a)->flags)
#define pthread_create(a, b, c, d) ((b) ? \
        thr_create(((pthread_attr_t *)(b))->base, \
                ((pthread_attr_t*)(b))->size, c, d, \
                ((pthread_attr_t *)(b))->flags, a) : \
        thr_create(NULL, 0, c, d, THR_BOUND, a))
#define pthread_join(a, b)              thr_join(a, NULL, b)

/* Straightforward translations for mutexes and conditions. */
#define pthread_mutex_init(a, b)        mutex_init(a, USYNC_THREAD, 0)
#define pthread_mutex_destroy           mutex_destroy
#define pthread_mutex_lock              mutex_lock
#define pthread_mutex_unlock            mutex_unlock
#define pthread_mutex_trylock           mutex_trylock
#define pthread_cond_init(a, b)         cond_init(a, USYNC_THREAD, 0)
#define pthread_cond_destroy            cond_destroy
#define pthread_cond_wait               cond_wait
#define pthread_cond_signal             cond_signal
#define pthread_cond_broadcast          cond_broadcast

/* Doing things once. */
#define pthread_once(once, func) { \
        if (!(once)->flag) { \
                pthread_mutex_lock(&(once)->lock); \
                if (!(once)->flag) { \
                        (*(func))(); \
                        (once)->flag = 1; \
                } \
                pthread_mutex_unlock(&(once)->lock); \
        } \
```

```
}

/* Thread-specific data. */
#define pthread_key_create              thr_keycreate
#define pthread_setspecific             thr_setspecific
static void *pthread_getspecific(pthread_key_t key) {
    void *value;

    return (thr_getspecific(key, &value) == 0) ? value : NULL;
}                                                                    70

/* gethostbyname_r.  This is a mess, and should be fixed in MIT pthreads,
 * but this is what we're going to do for the moment. */
#include <netdb.h>

struct hostent_answer { struct hostent hostent; char buf[2048]; };
static struct hostent *pthread_gethostbyname_r(const char *hostname,
                                               struct hostent_answer *result) {
    int err;
                                                                     80
    return gethostbyname_r(hostname, &result->hostent,
                           result->buf, sizeof(result->buf), &err);
}
#define gethostbyname_r pthread_gethostbyname_r
```

## A.3.5 afs.c

```
/*
 * $Id: afs.c,v 1.10 94/06/07 17:21:36 miki Exp $
 *
 * Copyright (c) 1990,1992 by the Massachusetts Institute of Technology.
 */

static char *rcsid = "$Id:  afs.c,v 1.10 94/06/07 17:21:36 miki Exp $";

#include "attach.h"
                                                                     10
#ifdef AFS

#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/file.h>
                                                                     20
#include <krb.h>

extern char *krb_realmofhost();    /* <krb.h> doesn't declare this */

/* Flags to afs_auth_internal() */
#define AFSAUTH_DOAUTH     1
```

```
#define AFSAUTH_CELL        2
#define AFSAUTH_DOZEPHYR    4

#ifdef _STDC_                                                          30
static int afs_auth_internal(char * errorname, char * afs_pathname,
                             struct in_addr hostaddr[], int authflags,
                             struct flags *flags);
#else
static int afs_auth_internal();
#endif


/*
 * The current implementation of AFS attaches is to make a symbolic    40
 * link.  This is NOT GUARANTEED to always be the case.
 */

int afs_attach(at, mopt, errorout, flags)
        struct _attachtab *at;
        struct mntopts  *mopt;
        int errorout;
        struct flags *flags;
{
        struct  stat    statbuf;                                       50
        char    buf[BUFSIZ];
        int     len;
        int     afs_auth_flags = 0;

        if ((at->mode != 'n') && flags->do_nfsid)
                afs_auth_flags |= AFSAUTH_DOAUTH;

        if (flags->use_zephyr)
                afs_auth_flags |= AFSAUTH_DOZEPHYR;
                                                                       60
        if ((afs_auth_flags & (AFSAUTH_DOAUTH | AFSAUTH_DOZEPHYR)) &&
           afs_auth_internal(at->hesiodname, at->hostdir,
                             at->hostaddr, afs_auth_flags, flags) == FAILURE)
                return(FAILURE);

        if (flags->debug_flag)
                printf("lstating %s...\n", at->hostdir);
        seteuid(flags->owner_uid);
        if (stat(at->hostdir, &statbuf)) {
                if (errno == ENOENT)                                   70
                        fprintf(stderr, "%s:  %s does not exist\n",
                                at->hesiodname, at->hostdir);
                else
                        perror(at->hostdir);
                flags->status = ERR_ATTACHNOFILSYS;
                seteuid(config.effective_uid);
                return(FAILURE);
        }
        if ((statbuf.st_mode & S_IFMT) != S_IFDIR) {
                fprintf(stderr, "%s:  %s is not a directory\n",        80
```

```
                        at->hesiodname, at->hostdir);
                seteuid(config.effective_uid);

                flags->status = ERR_ATTACHNOFILSYS;
                return(FAILURE);
        }

        if (flags->debug_flag)
                printf("lstating %s....\n", at->mntpt);
        if (!lstat(at->mntpt, &statbuf)) {                                    90
                seteuid(config.effective_uid);
                if ((statbuf.st_mode & S_IFMT) == S_IFLNK) {
                        len = readlink(at->mntpt, buf, sizeof(buf));
                        buf[len] = '\0';
                        (void) strcpy(buf, path_canon(buf));
                        if (!strcmp(buf, path_canon(at->hostdir))) {
                                if (!flags->print_path && flags->verbose)
                                   fprintf(stderr,
                                        "%s:  %s already sym linked%s\n",
                                        at->hesiodname, at->hostdir,          100
                                        (afs_auth_flags & AFSAUTH_DOAUTH) ?
                                        ", authenticating..." : "");
                                if (config.keep_mount)
                                        at->flags |= FLAG_PERMANENT;
                                return(SUCCESS);
                        } else {
                                if (config.keep_mount) {
                                        fprintf(stderr,
        "%s:  Couldn't attach; symlink from %s to %s already exists.\n",
                        at->hesiodname, at->mntpt, buf);                      110
                                        flags->status = ERR_ATTACHINUSE;
                                        return(FAILURE);
                                }
                                if (
#if defined(_AIX) && (AIXV==12)
                                   /* AIX 1.x */
                                   rmslink(at->mntpt) == -1 &&
#endif
                                   unlink(at->mntpt) == -1)
                                {                                             120
                                        fprintf(stderr,
        "%s:  Couldn't remove existing symlink from %s to %s:  %s\n",
                        at->hesiodname, at->mntpt, buf, strerror(errno));
                                        flags->status = ERR_ATTACHINUSE;
                                        return(FAILURE);
                                }
                        }
                } else if ((statbuf.st_mode & S_IFMT) == S_IFDIR) {
                        if (rmdir(at->mntpt)) {
                                if (errno == ENOTEMPTY)                       130
                                        fprintf(stderr,
                                "%s:  %s is a non-empty directory.\n",
                                                at->hesiodname, at->mntpt);
                                else
```

```
                                        fprintf(stderr,
                                                "%s:  Couldn't rmdir %s (%s)\n",
                                                at->hesiodname, at->mntpt,
                                                strerror(errno));
                                        flags->status = ERR_ATTACHINUSE;
                                        return(FAILURE);                                140
                                }
                        } else {
                                fprintf(stderr,
                                        "%s:  Couldn't attach; %s already exists.\n",
                                        at->hesiodname, at->mntpt);
                                flags->status = ERR_ATTACHINUSE;
                                return(FAILURE);
                        }
                }
                seteuid(config.effective_uid);                                          150

                /*
                 * Note: we do our own path canonicalization here, since
                 * we have to check the sym link first.
                 */
                (void) strcpy(at->mntpt, path_canon(at->mntpt));
                if (flags->debug_flag)
                        printf("Mountpoint canonicalized as:  %s\n", at->mntpt);

                if (!flags->override && !check_mountpt(at->mntpt, at->fs->type,         160
                                                       flags)) {
                        flags->status = ERR_ATTACHBADMNTPT;
                        return(FAILURE);
                }

                if (flags->debug_flag)
                        printf("symlinking %s to %s\n", at->hostdir, at->mntpt);
                if (symlink(at->hostdir, at->mntpt) < 0) {
                        fprintf(stderr, "%s:  ", at->hesiodname);
                        perror(at->hostdir);                                            170
/*                      flags->status = ERR_; */
                        return(FAILURE);
                }

                return (SUCCESS);
}

/*
 * afs_auth_internal --- authenticate oneself to the afs system
 *                                                                                      180
 * Actually, it also does the zephyr subscriptions, but that's because
 * aklog does all of that stuff.
 */
static int afs_auth_internal(errorname, afs_pathname, hostlist, authflags,
                             flags)
        char    *errorname;
        char    *afs_pathname;  /* For future expansion */
        struct in_addr hostlist[];
```

```
        int     authflags;
        struct flags *flags;                                    190
{
#ifdef POSIX
        int     waitb;
#else
        union wait      waitb;
#endif
        int     error_ret;
        int     fds[2];
        FILE    *f;
        char    buff[512];                                      200
        int     host_idx = 0;
        pid_t   pid;

        if (flags->debug_flag)
                printf("performing an %s %s %s -hosts -zsubs %s\n",
                    config.aklog_fn,
                    authflags & AFSAUTH_CELL ? "-cell" : "-path",
                    afs_pathname,
                    authflags & AFSAUTH_DOAUTH ? "" : "-noauth");
                                                                210
        if (pipe(fds)) {
                perror("afs_auth:  pipe");
                return(FAILURE);
        }

        switch(pid = fork()) {
        case -1:
                close(fds[0]);
                close(fds[1]);
                perror("vfork:  to aklog");                     220
                flags->status = ERR_AUTHFAIL;
                return(FAILURE);
        case 0:
                if (!flags->debug_flag) {
                        close(0);
                        open("/dev/null", O_RDWR, 0644);
                }

                close(fds[0]);
                dup2(fds[1], 1);                                230
                close(fds[1]);
                setuid(flags->owner_uid);
                execl(config.aklog_fn, AKLOG_SHORTNAME,
                    (authflags & AFSAUTH_CELL) ? "-cell" : "-path",
                    afs_pathname, "-hosts", "-zsubs",
                    (authflags & AFSAUTH_DOAUTH) ? 0 : "-noauth", 0);
                perror(config.aklog_fn);
                exit(1);
                /*NOTREACHED*/
        default:                                                240
                close(fds[1]);
                if ((f = fdopen(fds[0], "r")) == NULL) {
```

```
                        perror("fdopen:  to aklog output");
                        return(FAILURE);
                }
                while (fgets(buff, sizeof(buff), f)) {
                        char    *cp;
                        int     len;

                        if (flags->debug_flag)                                  250
                                fputs(buff, stdout);
                        if (!(cp = strchr(buff, ':')))
                                continue;
                        *cp = '\0';
#ifdef ZEPHYR
                        if (!strcmp(buff, "zsub") &&
                           (authflags & AFSAUTH_DOZEPHYR)) {
                                cp++;
                                while (*cp && isspace(*cp))
                                        cp++;                                   260
                                for (len=strlen(cp)-1;
                                    len>=0 && !isprint(cp[len])
                                    ; len--)
                                        cp[len] = '\0';
                                zephyr_addsub(cp, flags);
                        }
#endif
                        if (hostlist && !strcmp(buff, "host")) {
                                cp++;
                                while (*cp && isspace(*cp))                     270
                                        cp++;
                                for (len=strlen(cp)-1;
                                    len>=0 && !isprint(cp[len])
                                    ; len--)
                                        cp[len] = '\0';
                                hostlist[host_idx++].s_addr = inet_addr(cp);
                        }
                }
                fclose(f);
                if (waitpid(pid, &waitb, 0) < 0) {                              280
                        perror("wait:  for aklog");
                        flags->status = ERR_AUTHFAIL;
                        return(FAILURE);
                }
        }

#ifdef POSIX
        error_ret = (WIFEXITED(waitb)) ? WEXITSTATUS(waitb) : 1;
#else
        error_ret = waitb.w_retcode;                                            290
#endif
        if (error_ret && (authflags & AFSAUTH_DOAUTH)) {
                flags->status = ERR_AUTHFAIL;
                return (FAILURE);
        }
        return(SUCCESS);
```

```
}

int afs_auth(hesname, afsdir, flags)
        char *hesname, *afsdir;                                          300
        struct flags *flags;
{
   return(afs_auth_internal(hesname, afsdir, 0, AFSAUTH_DOAUTH, flags));
}

/*
 * Parsing of explicit AFS file types
 */
int afs_explicit(name, hesptr, hesline, flags)
        char *name, **hesptr, *hesline;                                  310
        struct flags *flags;
{
  char *dir;
  char newmntpt[BUFSIZ];

  if (*name != '/')
    {
     fprintf(stderr, "%s:  Illegal explicit definition \"%s\" for type %s\n",
             config.progname, name, flags->filsys_type);
     return -1;                                                          320
    }

  dir = strrchr(name, '/');
  (void) strcpy(newmntpt, config.afs_mount_dir);
  (void) strcat(newmntpt, dir);

  sprintf(hesline, "AFS %s %c %s", name, flags->override_mode ?
          flags->override_mode : 'w', flags->mntpt ? flags->mntpt : newmntpt);
  hesptr[0] = hesline;
  hesptr[1] = 0;                                                         330
  return 0;
}

int afs_detach(at, list, flags)
        struct _attachtab *at;
        struct attach_list *list;
        struct flags *flags;
{
   if(at->flags & FLAG_PERMANENT)
        return SUCCESS;                                                  340

   if (
#if defined(_AIX) && (AIXV==12)
        /* AIX 1.x */
        rmslink(at->mntpt) == -1 &&
#endif
        unlink(at->mntpt) == -1)
        {
           if(errno == ENOENT)
                {                                                        350
```

```
                fprintf(stderr, "%s:  filesystem %s already detached\n",
                        config.progname, at->hesiodname);
                return SUCCESS;
            }
        fprintf(stderr, "%s:  detach of filesystem %s failed, unable to remove mountpoint\n\te
                config.progname, at->hesiodname, strerror(errno));
        /* Set error_status? */
        return FAILURE;
    }
  return SUCCESS;                                                    360
}

int afs_auth_to_cell(cell, flags)
    char *cell;
    struct flags *flags;
{
  return(afs_auth_internal(cell, cell, 0, AFSAUTH_DOAUTH | AFSAUTH_CELL,
                           flags));
}
                                                                     370
int afs_zinit(hesname, afsdir, flags)
    char *hesname, *afsdir;
    struct flags *flags;
{
  return(afs_auth_internal(hesname, afsdir, 0, AFSAUTH_DOZEPHYR, flags));
}
#endif /* AFS */
```

## A.3.6 attach.c

```
/*      Created by:         Robert French
 *
 *      $Source: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/attach.c,v $
 *      $Author: miki $
 *
 *      Copyright (c) 1988 by the Massachusetts Institute of Technology.
 */

#ifndef lint
static char rcsid_attach_c[] = "$Header:  /afs/dev.mit.edu/project/release/source/src/athena/bin/
#endif

#include "attach.h"
#include <signal.h>

static void print_status _P((char *msg, struct flags *flags));

void *
attach_threadfunc(arg)
  void *arg;                                                         20
{
```

```
    struct flags *flags = (struct flags *) arg;

    flags->status = ERR_NONE;
    attach(flags->fsname, flags);
    return flags;
}

/*
 * Attempt to attach a filesystem.  Lookup the name with Hesiod as
 * necessary, and try all possible results in order.  Also handles
 * the explicit flag for explicitly defined hostname/directory
 * arguments.  Branches to NFS or RVD attach routines as necessary.
 */

int
attach(name, flags)
    char *name;
    struct flags *flags;
{
    struct attach_list      list = ATTACH_LIST_INITIALIZER;
    struct _attachtab       at;
    struct _attachtab       *atp;
    struct fslock           fslock;
    int                     result;
    int                     i;
    int                     print_wait;
    extern int              caught_signal;
    char                    *hesvec[32];
    char                    hesline[BUFSIZ];

    print_status("beginning", flags);
    result = build_hesiod_line(name, hesvec, 32, hesline, flags);
    print_status("resolved", flags);

    if (result != 0 || !*hesvec) {
        flags->status = ERR_ATTACHBADFILSYS;
        return FAILURE;
    }

    if (flags->lookup || flags->debug_flag) {
        printf("%s resolves to:\n", name);
        for (i=0;hesvec[i];i++)
            printf("%s\n", hesvec[i]);
        putchar('\n');
    }

    if (flags->lookup)
            return SUCCESS;

    print_wait = 2;
retry:
    print_status("Before initial get", flags);
    lock_attachtab(&list, flags);
    get_attachtab(&list, flags);
```

```
print_status("After initial get", flags);
if (atp = attachtab_lookup(name, &list, flags)) {
        switch (atp->status) {
        case STATUS_ATTACHING:
          if (!flags->force && really_in_use(name, flags)) {                    80
                unlock_attachtab(&list, flags);
                free_attachtab(&list, flags);
                if (flags->print_path) {
                        sleep(print_wait);  /* Wait a few seconds */
                        if (print_wait < 30)
                                print_wait *= 2;
                        goto retry;
                }
                fprintf(stderr,
                        "%s:  Filesystem \"%s\" already being attached by another process\n",
                        config.progname, name);
                flags->status = ERR_ATTACHINUSE;
                return FAILURE;
          }
          attachtab_delete(atp, &list, flags);
          put_attachtab(&list, flags);
          break;
        case STATUS_DETACHING:
          if (!flags->force && really_in_use(name, flags)) {
                fprintf(stderr, "%s:  Filesystem \"%s\" is being detached by another process\n",
                        config.progname, name);
                unlock_attachtab(&list, flags);
                free_attachtab(&list, flags);
                flags->status = ERR_ATTACHINUSE;
                return FAILURE;
          }
          attachtab_delete(atp, &list, flags);
          put_attachtab(&list, flags);
          break;
        case STATUS_ATTACHED:                                                   110
          if (flags->force)
                break;
          if (flags->lock_filesystem)
                  atp->flags |= FLAG_LOCKED;
          if (config.owner_list)
                  add_an_owner(atp,flags->owner_uid);
          if (config.owner_list || flags->lock_filesystem)
                  put_attachtab(&list, flags);
          unlock_attachtab(&list, flags);
                                                                                120
          if (atp->fs->type == TYPE_MUL)
                return mul_attach(atp, NULL, 0, flags);

          if (flags->print_path) {
                sprintf(flags->outbuf, "%s\n", atp->mntpt);
                flags->output = flags->outbuf;
          } else if(flags->verbose) {
                sprintf(flags->outbuf,
                        "%s:  Filesystem \"%s\" is already attached",
```

```
                    config.progname, name);
                flags->output = flags->outbuf;
        }
#ifdef NFS
        if (flags->map_anyway && atp->mode != 'n'
            && atp->fs->type == TYPE_NFS) {
            int ret;

            if (flags->verbose && !flags->print_path)
                strcat(flags->outbuf, " (mapping)\n");

            ret = nfsid(atp->host, atp->hostaddr[0], MOUNTPROC_KUIDMAP,
                        1, name, 1, config.real_uid, flags);
            if(atp->mode != 'm')
              return ret;
            if (ret == FAILURE)
              {
                flags->status = ERR_NONE;
                clear_errored(atp->hostaddr[0]);
              }
            return SUCCESS;
        }
#endif
#ifdef AFS
        if (flags->map_anyway && atp->mode != 'n' &&
            atp->fs->type == TYPE_AFS) {
                if (flags->verbose && !flags->print_path)
                        strcat(flags->outbuf, " (authenticating)\n");
                return(afs_auth(atp->hesiodname, atp->hostdir, flags));
        }
#endif
        strcat(flags->outbuf, "\n");
        if (flags->verbose && !flags->print_path)
            putchar('\n');
        /* No error code set on already attached */
        free_attachtab(&list, flags);
        return SUCCESS;
      }
  }

  /* Note: attachtab is still locked at this point */

  /* We don't really care about ANY of the values besides status */
  at.status = STATUS_ATTACHING;
  at.explicit = flags->explicit;
  at.fs = NULL;
  strcpy(at.hesiodname, name);
  strcpy(at.host, "?");
  strcpy(at.hostdir, "?");
  memset((char *)&at.hostaddr, 0, sizeof(at.hostaddr));
  at.rmdir = 0;
  at.drivenum = 0;
  at.mode = 'r';
  at.flags = 0;
```

```
at.nowners = 1;
at.owners[0] = flags->owner_uid;
strcpy(at.mntpt, "?");

/*
 * Mark the filsys as "being attached".
 */                                                          190
start_critical_code();              /* Don't let us be interrupted */
mark_in_use(name, &fslock, flags);
attachtab_append(&at, &list, flags);
put_attachtab(&list, flags);
unlock_attachtab(&list, flags);
print_status("After initial write", flags);

for (i=0;hesvec[i];i++) {
        if (flags->debug_flag)
           printf("Processing line %s\n", hesvec[i]);             200
        if (caught_signal) {
                if (flags->debug_flag)
                        printf("Caught signal; cleaning up attachtab....\n");
                free_attachtab(&list, flags);
                lock_attachtab(&list, flags);
                get_attachtab(&list, flags);
                attachtab_delete(attachtab_lookup(at.hesiodname, &list, flags),
                                &list, flags);
                put_attachtab(&list, flags);
                unlock_attachtab(&list, flags);                 210
                free_attachtab(&list, flags);
                terminate_program();
        }
        /*
         * Note try_attach will change attachtab appropriately if
         * successful.
         */
        if (try_attach(name, hesvec[i], !hesvec[i+1], &list,
                        flags) == SUCCESS) {
                free_attachtab(&list, flags);                   220
                mark_in_use(NULL, &fslock, flags);
                end_critical_code();
                return SUCCESS;
        }
}
free_attachtab(&list, flags);

if (flags->status == ERR_ATTACHNOTALLOWED)
        fprintf(stderr, "%s:  You are not allowed to attach %s.\n",
                config.progname, name);                         230

if (flags->status == ERR_ATTACHBADMNTPT)
        fprintf(stderr,
                "%s:  You are not allowed to attach a filesystem here\n",
                config.progname);
/*
 * We've failed -- delete the partial entry and unlock the in-use file.
```

```
     */
    lock_attachtab(&list, flags);
    get_attachtab(&list, flags);                                          240
    attachtab_delete(attachtab_lookup(at.hesiodname, &list, flags), &list,
                     flags);
    put_attachtab(&list, flags);
    mark_in_use(NULL, &fslock, flags);
    unlock_attachtab(&list, flags);
    free_attachtab(&list, flags);
    end_critical_code();

    /* Assume error code already set by whatever made us fail */
    return FAILURE;                                                       250
}

/*
 * Given a Hesiod line and a filesystem name, try to attach it.  If
 * successful, change the attachtab accordingly.
 */

try_attach(name, hesline, errorout, list, flags)
    char *name, *hesline;
    int errorout;                                                         260
    struct attach_list *list;
    struct flags *flags;
{
    struct _attachtab at, *atp;
    int status;
    int    attach_suid;
#ifdef ZEPHYR
    char instbfr[BUFSIZ];
#endif
    struct mntopts          mopt;                                         270
    char            *default_options;

    print_status("trying", flags);
    if (parse_hes(hesline, &at, name)) {
            flags->status = ERR_BADFSDSC;
            return FAILURE;
    }
    if (flags->filsys_type && *flags->filsys_type &&
        strcasecmp(flags->filsys_type, at.fs->name) != 0) {
            flags->status = ERR_ATTACHBADFILSYS;                          280
            return FAILURE;
    }
    if (!flags->override && !allow_filsys(name, at.fs->type, flags)) {
            flags->status = ERR_ATTACHNOTALLOWED;
            return FAILURE;
    }

    at.status = STATUS_ATTACHING;
    at.explicit = flags->explicit;
    strcpy(at.hesiodname, name);                                          290
    add_an_owner(&at, flags->owner_uid);
```

```
if (flags->mntpt)
        strcpy(at.mntpt, flags->mntpt);

/*
 * Note if a filesystem does nothave AT_FS_MNTPT_CANON as a property,
 * it must also somehow call check_mntpt, if it wants mountpoint
 * checking to happen at all.
 */                                                                    300
if (at.fs->flags & AT_FS_MNTPT_CANON) {
        char *path;

        /* Perform path canonicalization */
        if ((path = path_canon(at.mntpt)) == NULL) {
                fprintf(stderr, "%s:  Cannot get information about the mountpoint %s\n",
                        at.hesiodname, at.mntpt);
                flags->status = ERR_SOMETHING;
                return FAILURE;
        }                                                              310
        strcpy(at.mntpt, path);
        if (flags->debug_flag)
                printf("Mountpoint canonicalized as:  %s\n", at.mntpt);
        if (!flags->override && !check_mountpt(at.mntpt, at.fs->type,
                                                flags)) {
                flags->status = ERR_ATTACHBADMNTPT;
                return FAILURE;
        }
}
                                                                       320
if (at.fs->flags & AT_FS_MNTPT
      && (atp=attachtab_lookup_mntpt(at.mntpt, list, flags))) {
        fprintf(stderr,"%s:  Filesystem %s is already mounted on %s\n",
                at.hesiodname, atp->hesiodname, at.mntpt);
        flags->status = ERR_ATTACHDIRINUSE;
        return FAILURE;
}

if (flags->override_mode)
      at.mode = flags->override_mode;                                  330

if (flags->override_suid == -1)
        attach_suid = !nosetuid_filsys(name, at.fs->type, flags);
else
        attach_suid = flags->override_suid;
at.flags = (attach_suid ? 0 : FLAG_NOSETUID) +
        (flags->lock_filesystem ? FLAG_LOCKED : 0);

/* Prepare mount options structure */
memset(&mopt, 0, sizeof(mopt));                                        340
mopt.type = at.fs->mount_type;

/* Read in default options */
default_options = filsys_options(at.hesiodname, at.fs->type, flags);
add_options(&mopt, "soft");             /* Soft by default */
```

```
if (flags->mount_options && *flags->mount_options)
        add_options(&mopt, flags->mount_options);
if (default_options && *default_options)
        add_options(&mopt, default_options);
                                                                    350
if (at.mode == 'r')
        add_options(&mopt, "ro");
if (at.flags & FLAG_NOSETUID)
        add_options(&mopt, "nosuid");

print_status("before filsys routine", flags);
if (at.fs->attach) {
        if (at.fs->flags & AT_FS_MNTPT) {
                if (make_mntpt(&at, flags) == FAILURE) {
                        rm_mntpt(&at, flags);                       360
                        return FAILURE;
                }
        }
        status = (at.fs->attach)(&at, &mopt, errorout, flags);
} else {
        fprintf(stderr,
                "%s:  Can't attach filesystem type \"%s\"\n",
                config.progname, at.fs->name);
        flags->status = ERR_FATAL;
        return FAILURE;                                             370
}
print_status("after filsys routine", flags);

if (status == SUCCESS) {
      char    tmp[BUFSIZ];
      if (at.fs->flags & AT_FS_REMOTE)
              sprintf(tmp, "%s:%s", at.host, at.hostdir);
      else
              strcpy(tmp, at.hostdir);
      if (flags->verbose) {                                         380
              if(at.fs->type == TYPE_AFS)
                sprintf(flags->outbuf,
                        "%s:  %s linked to %s for filesystem %s\n",
                        config.progname, tmp, at.mntpt, at.hesiodname);
              else
                sprintf(flags->outbuf,
                        "%s:  filesystem %s (%s) mounted on %s (%s)\n",
                        config.progname, at.hesiodname, tmp,
                        at.mntpt, (mopt.flags & M_RDONLY) ? "read-only" :
                        "read-write");                              390
              flags->output = flags->outbuf;
      }
      if (flags->print_path) {
              sprintf(flags->outbuf, "%s\n", at.mntpt);
              flags->output = flags->outbuf;
      }
      at.status = STATUS_ATTACHED;
      lock_attachtab(list, flags);
      get_attachtab(list, flags);
```

```
        attachtab_replace(&at, list, flags);                                    400
        put_attachtab(list, flags);
        unlock_attachtab(list, flags);
        /*
         * Do Zephyr stuff as necessary
         */
#ifdef ZEPHYR
        if (flags->use_zephyr && at.fs->flags & AT_FS_REMOTE) {
                if(at.fs->type == TYPE_AFS) {
                        afs_zinit(at.hesiodname, at.hostdir, flags);
                } else {                                                        410
                        sprintf(instbfr, "%s:%s", at.host, at.hostdir);
                        zephyr_addsub(instbfr, flags);
                        zephyr_addsub(at.host, flags);
                }
        }
#endif
        free_attachtab(list, flags);
    } else
            if (at.fs->flags & AT_FS_MNTPT)
                    rm_mntpt(&at, flags);                                       420
    return (status);
}


int attach_print(host, print_banner, flags)
    char *host;
    int print_banner;
    struct flags *flags;
{
    static char *attach_list_format = "%-22s %-22s %c%-18s%s\n";                430
    struct attach_list list = ATTACH_LIST_INITIALIZER;
    struct _attachtab *atp;
    char optstr[40];
    int bad = 0;
    int i;

    lock_attachtab(&list, flags);
    get_attachtab(&list, flags);
    unlock_attachtab(&list, flags);
    atp = list.attachtab_first;                                                 440
    if (!atp) {
        printf("No filesystems currently attached.\n");
        free_attachtab(&list, flags);
        return(ERR_NONE);
    }
    if (print_banner) {
        printf(attach_list_format, "filesystem", "mountpoint",
            ' ', "user", "mode");
        printf(attach_list_format, "----------", "----------",
            ' ', "----", "----");                                               450
    }
    while (atp) {
        optstr[0] = atp->mode;
```

```
        optstr[1] = '\0';
        if (atp->flags & FLAG_NOSETUID)
            strcat(optstr, ",nosuid");
        if (atp->flags & FLAG_LOCKED)
            strcat(optstr, ",locked");
        if (atp->flags & FLAG_PERMANENT)
            strcat(optstr, ",perm");                                    460
        if (host) {
            bad = 1;
            for (i=0; i<MAXHOSTS && atp->hostaddr[i].s_addr; i++) {
                if (host_compare(host, (char *) inet_ntoa(atp->hostaddr[i]),
                                 flags)) {
                    bad = 0;
                    break;
                }
            }
        }                                                               470
        if (!bad && atp->status == STATUS_ATTACHED) {
            printf(attach_list_format, atp->hesiodname,
                   (atp->fs->type&TYPE_MUL) ? "-" : atp->mntpt,
                   atp->flags & FLAG_ANYONE ? '*' : ' ',
                   ownerlist(atp), optstr);
        }
        atp = atp->next;
    }
    free_attachtab(&list, flags);
    return (ERR_NONE);                                                  480
}

static void print_status(msg, flags)
    char *msg;
    struct flags *flags;
{
#ifdef TIMING
    struct timeval tv;
    struct timezone zone;
                                                                        490
    gettimeofday(&tv, &zone);
    printf("%12s %10ld %10ld %s\n", flags->fsname, tv.tv_sec, tv.tv_usec, msg);
#endif
}
```

## A.3.7 config.c

```
/*
 *      $Source: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/config.c,v $
 *      $Author: miki $
 *      $Header: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/config.c,v 1.7 94/06/07 17:2
 */

#ifndef lint
static char *rcsid_config_c = "$Header:  /afs/dev.mit.edu/project/release/source/src/athena/bin/attach.
```

```
#endif

#include "attach.h"
#include "pwd.h"
#include <string.h>
#include <sys/types.h>
#include <regex.h>


#define TOKSEP " \t\r\n"

/*
 * Declare the filesystem tables for nosetuid and allowable mountings
 */
struct filesys_tab {
        struct _tab {
                char    *regexp;
                int     fs_type;
                int     explicit; /* 0 = don't care, 1 = explicit */
                                  /* only, -1 = no explicit only */
                char    *tab_data;
        } tab[MAXFILTAB];
        int     idx;            /* Index to next free entry */
} nosuidtab, allowtab, goodmntpt, fsdb, optionsdb;

static char     data_t[] = "#!TRUE";

struct uid_tab {
        int     tab[MAXTRUIDTAB];
        int     idx;
} trusted_uid;                  /* trusted uid table */

/*
 * Define the keyword table
 */
static void parse_boolean _P((char *keyword, char *argument, char *buff,
                              void *variable, int arg));
static void parse_string _P((char *keyword, char *argument, char *buff,
                             void *variable, int arg));
static void parse_fslist _P((char *keyword, char *argument, char *buff,
                             void *variable, int arg));
static void parse_fslista _P((char *keyword, char *argument, char *buff,
                              void *variable, int arg));
static void parse_uidlist _P((char *keyword, char *argument, char *buff,
                              void *variable, int arg));
static int get_fstype _P((char **cpp, int *explicit));

struct key_def {
        char    *key;
        void    (*proc) _P((char *, char *, char *, void *, int));
        void    *variable;
        int     arg;
} keyword_list[] = {
        {"verbose",     parse_boolean,  &config.verbose,        1 },
```

```
        {"debug",       parse_boolean,    &config.debug_flag,       1 },
        {"ownercheck",  parse_boolean,    &config.owner_check,      1 },
        {"ownerlist",   parse_boolean,    &config.owner_list,       1 },
        {"keep-mount",  parse_boolean,    &config.keep_mount,       1 },
        {"explicit-mntpt",parse_boolean,&config.exp_mntpt,          1 },
        {"explicit",    parse_boolean,    &config.exp_allow,        1 },

        {"nfs-root-hack", parse_boolean,&config.nfs_root_hack,      1 },
        {"nfs-mount-dir", parse_string, &config.nfs_mount_dir,      0 },

        {"attachtab",   parse_string,     &config.attachtab_fn,     0 },
        {"mtab",        parse_string,     &config.mtab_fn,          0 },
#ifdef AFS
    {"aklog",     parse_string,   &config.aklog_fn,               0 },
        {"afs-mount-dir", parse_string,   &config.afs_mount_dir,    0 },
#endif
        {"fsck",        parse_string,     &config.fsck_fn,          0 },

        {"trusted",     parse_uidlist,    &trusted_uid,             0 },
        {"nosetuid",    parse_fslist,     &nosuidtab,               1 },
        {"nosuid",      parse_fslist,     &nosuidtab,               1 },
        {"setuid",      parse_fslist,     &nosuidtab,               0 },
        {"suid",        parse_fslist,     &nosuidtab,               0 },
        {"allow",       parse_fslist,     &allowtab,                1 },
        {"noallow",     parse_fslist,     &allowtab,                0 },
        {"mountpoint",  parse_fslist,     &goodmntpt,               1 },
        {"nomountpoint",parse_fslist,     &goodmntpt,               0 },

        {"filesystem",  parse_fslista,    &fsdb,                    0 },
        {"options",     parse_fslista,    &optionsdb,               0 },
        {NULL,          NULL,             NULL,                     0 }
};

static void config_abort()
{
        fprintf(stderr, " Aborting...\n");
        exit(ERR_BADCONF);
}

read_config_file(config_file_name)
        const char *config_file_name;
{
        register FILE   *f;
        static char     buff[256];
        char            *cp;
        register char   *keyword;
        char            *argument;
        struct key_def  *kp;

        allowtab.tab[0].regexp = NULL;
        nosuidtab.tab[0].regexp = NULL;
        goodmntpt.tab[0].regexp = NULL;
        fsdb.tab[0].regexp = NULL;
        trusted_uid.tab[0] = -1;
```

```
        if (config.debug_flag)
                printf("Reading configuration file:  %s\n", config_file_name);
        if (!(f = fopen(config_file_name, "r"))) {
                if (config.debug_flag)                                          120
                        printf("Couldn't read conf file, continuing...\n");
                return(SUCCESS);
        }
        while (fgets(buff, sizeof(buff), f)) {
                cp = buff+strlen(buff)-1;
                while (cp >= buff && *cp < 032)
                        *cp-- = '\0';
                cp = buff;
                while (*cp && isspace(*cp)) cp++;
                if (!*cp || *cp == '#')                                         130
                        continue;
                keyword = strtok(cp, TOKSEP);
                argument = strtok(NULL, TOKSEP);
                for (kp=keyword_list; kp->key; kp++) {
                        if (!strcmp(kp->key, keyword)) {
                                (kp->proc)(keyword,argument,buff,
                                        kp->variable, kp->arg);
                                break;
                        }
                }                                                               140
                if (!kp->key) {
                        if (config.debug_flag)
                fprintf(stderr, "%s:  bad keyword in config file\n",
                                        keyword);
                }
        }
        return(SUCCESS);
}

static void parse_boolean(keyword, argument, buff, variable, arg)               150
        char    *keyword, *argument, *buff;
        void    *variable;
        int     arg;
{
        if (argument && !strcmp(argument, "on"))
                * (int *) variable = 1;
        else if (argument && !strcmp(argument, "off"))
                * (int *) variable = 0;
        else {
                if (arg == -1)                                                  160
                        if (argument)
                          fprintf(stderr,
                                "%s:  Argument to %s must be on or off!\n",
                                argument, keyword);
                        else
                          fprintf(stderr, "%s:  Argument required!\n", keyword);
                else
                        * (int *) variable = arg;               /* Default */
        }
}                                                                               170
```

```
static void parse_string(keyword, argument, buff, variable, arg)
        char    *keyword, *argument, *buff;
        void    *variable;
        int     arg;
{
        if (!argument || !*argument) {
                fprintf(stderr,
                        "%s:  missing argument in config file!\n",
                        keyword);                                          180
                config_abort();
        }
        if (* (char **) variable)
                free(* (char **) variable);
        * (char **) variable = strdup(argument);
}

static void parse_fslist(keyword, argument, buff, variable, set_sw)
        char    *keyword, *argument, *buff;
        void    *variable;                                                 190
        int     set_sw;
{
        register struct filesys_tab          *fslist;

        fslist = (struct filesys_tab *) variable;
        while (argument && *argument) {
                if (fslist->idx >= MAXFILTAB) {
                        fprintf(stderr, "Exceeded filesystem table for %s!\n",
                                keyword);
                        config_abort();                                    200
                }
                fslist->tab[fslist->idx].fs_type = get_fstype(&argument,
                                        &fslist->tab[fslist->idx].explicit);
                fslist->tab[fslist->idx].regexp = strdup(argument);
                fslist->tab[fslist->idx].tab_data = set_sw ? data_t : NULL;
                fslist->idx++;
                argument = strtok(NULL, TOKSEP);
        }
        fslist->tab[fslist->idx].regexp = NULL;
        fslist->tab[fslist->idx].tab_data = NULL;                          210
}

static void parse_fslista(keyword, argument, buff, variable, dummy)
        char    *keyword, *argument, *buff;
        void    *variable;
        int     dummy;
{
        register struct filesys_tab          *fslist;
        char    *cp;
                                                                           220
        fslist = (struct filesys_tab *) variable;
        if (fslist->idx >= MAXFILTAB) {
                fprintf(stderr, "Exceeded filesystem table for %s!\n",
                        keyword);
```

```
                config_abort();
        }
        fslist->tab[fslist->idx].fs_type = get_fstype(&argument,
                                        &fslist->tab[fslist->idx].explicit);
        fslist->tab[fslist->idx].regexp = strdup(argument);
        cp = strtok(NULL, "");                                          230
        while (cp && *cp)
                if (isspace(*cp))
                        cp++;
                else
                        break;

        if (cp && *cp) {
                fslist->tab[fslist->idx].tab_data = strdup(cp);
        } else {
                fprintf(stderr,                                         240
                        "%s:  missing filesystem definition in config file\n");
                config_abort();
        }
        fslist->idx++;

        fslist->tab[fslist->idx].regexp = NULL;
        fslist->tab[fslist->idx].tab_data = NULL;
}

/*                                                                      250
 * Parse a filesystem type specification
 *
 * Format:    {nfs,afs}:tytso
 *            {^ufs}:.*
 *
 */
static int get_fstype(cpp, explicit)
        char    **cpp;
        int     *explicit;
{                                                                       260
        register char   *cp, *ptrstart;
        register int    parsing, type;
        int     negate = 0;
        struct _fstypes *fs;
        int     exp = 0;

        if (explicit)
                *explicit = 0;
        cp = *cpp;
        if (!cp || *cp++ != '{')                                        270
                return(ALL_TYPES);
        if (*cp == '+') {
                exp = 1;
                cp++;
        } else if (*cp == '-') {
                exp = -1;
                cp++;
        }
```

```
        if (explicit)
                *explicit = exp;                                        280
        if (!strchr(cp, '}')) {
                fprintf(stderr,"Error in filesystem specification:\n{%s\n",
                        cp);
                config_abort();
        }
        if (*cp == '^') {
                negate++;
                cp++;
        }
        parsing = 1;                                                    290
        type = 0;
        while (parsing) {
                ptrstart = cp;
                if (!(cp = strchr(ptrstart,',' ))) {
                        cp = strchr(ptrstart,'}');
                        parsing = 0;
                }
                *cp++ = '\0';
                fs = get_fs(ptrstart);
                if (!fs) {                                              300
                        fprintf(stderr,
                                "%s:  Illegal filesystem type in config file\n",
                                ptrstart);
                        config_abort();
                }
                type += fs->type;
        }
        if (negate)
                type = ~type & ALL_TYPES;
        if (*cp == ':')                                                 310
                cp++;
        *cpp = cp;
        return(type);
}

static void parse_uidlist(keyword, argument, buff, variable, arg)
        char    *keyword, *argument, *buff;
        void    *variable;
        int     arg;
{                                                                       320
        register struct uid_tab         *ul;
        int     uid;
        struct passwd   *pw;

        ul = (struct uid_tab *) variable;
        while (argument && *argument) {
                if (ul->idx >= MAXTRUIDTAB) {
                        fprintf(stderr, "Exceeded user table for %s!\n",
                                keyword);
                        config_abort();                                 330
                }
                if (isnumber(argument))
```

```
                        uid = atoi(argument);
                else {
                        if ((pw = getpwnam(argument)))
                                uid = pw->pw_uid;
                        else {
                                if (config.debug_flag)
                                        fprintf(stderr,
                                        "Unknown user %s in config file\n",
                                                argument);
                                argument = strtok(NULL, TOKSEP);
                                continue;
                        }
                }
                ul->tab[ul->idx] = uid;
                ul->idx++;
                argument = strtok(NULL, TOKSEP);
        }
        ul->tab[ul->idx] = -1;
}

isnumber(s)
        register char   *s;
{
        register char   c;

        while (c = *s++)
                if (!isdigit(c))
                        return(0);
        return(1);
}

int nosetuid_filsys(name, type, flags)
        register char           *name;
        int                     type;
        struct flags            *flags;
{
        register struct _tab    *fp;
        int                     retval;
        char                    errbuf[100];
        regex_t                 reg;

        for (fp = nosuidtab.tab; fp->regexp; fp++) {
                if (fp->explicit && ((flags->explicit && fp->explicit == -1) ||
                                        (!flags->explicit && fp->explicit == 1)))
                        continue;
                if (!(fp->fs_type & type))
                        continue;

                if (retval=regcomp(&reg, fp->regexp, 0)) {
                        regerror(retval, &reg, errbuf, sizeof(errbuf));
                        fprintf(stderr, "Nosetuid config:  %s:  %s",
                                errbuf, fp->regexp);
                        return(config.default_suid);
                }
```

```
                if (regexec(&reg, name, 0, NULL, 0) == 0)
                        return(fp->tab_data ? 1 : 0);
        }
                                                                        390
        return(config.default_suid);
}

int allow_filsys(name, type, flags)
        register char           *name;
        int                     type;
        struct flags            *flags;
{
        register struct _tab    *fp;
        int                     retval;                                 400
        char                    errbuf[100];
        regex_t                 reg;


        for (fp = allowtab.tab; fp->regexp; fp++) {
                if (fp->explicit && ((flags->explicit && fp->explicit == -1) ||
                                        (!flags->explicit && fp->explicit == 1)))
                        continue;
                if (!(fp->fs_type & type))
                        continue;                                       410

                if (retval=regcomp(&reg, fp->regexp, 0)) {
                        regerror(retval, &reg, errbuf, sizeof(errbuf));
                        fprintf(stderr, "Allow config:  %s:  %s",
                                errbuf, fp->regexp);
                        return(1);
                }
                if (regexec(&reg, name, 0, NULL, 0) == 0)
                        return(fp->tab_data ? 1 : 0);
                                                                        420
        }
        return(1);              /* Default to allow filsystem to be mounted */
}

int check_mountpt(name, type, flags)
        register char           *name;
        int                     type;
        struct flags            *flags;
{
        register struct _tab    *fp;                                    430
        int                     retval;
        char                    errbuf[100];
        regex_t                 reg;


        for (fp = goodmntpt.tab; fp->regexp; fp++) {
                if (fp->explicit && ((flags->explicit && fp->explicit == -1) ||
                                        (!flags->explicit && fp->explicit == 1)))
                        continue;
                if (!(fp->fs_type & type))                              440
```

```
                        continue;


                if (retval=regcomp(&reg, fp->regexp, 0)) {
                        regerror(retval, &reg, errbuf, sizeof(errbuf));
                        fprintf(stderr, "Mountpoint config:  %s:  %s",
                                errbuf, fp->regexp);
                        return(1);
                }
                if (regexec(&reg, name, 0, NULL, 0) == 0)                    450
                        return(fp->tab_data ? 1 : 0);

        }
        return(1);                      /* Default to allow filsystem mounted */
                                        /* anywhere */
}

/*
 * Return true if uid is of a trusted user.
 */                                                                          460
int trusted_user(uid)
        int     uid;
{
        int     *ip;

        ip = trusted_uid.tab;
        while (*ip >= 0) {
                if (uid == *ip)
                        return(1);
                ip++;                                                        470
        }
        return(uid == 0);           /* Hard code root being trusted */
}

/*
 * Look up a filesystem name and return a "hesiod" entry
 */
int conf_filsys_resolve(name, hesptr)
        char    *name;
        char    **hesptr;                                                    480
{
        register struct  _tab     *fp;
        char    **hp;

        hp = hesptr;
        for (fp = fsdb.tab; fp->regexp; fp++) {
                if (!strcasecmp(fp->regexp, name))
                        *hp++ = strdup(fp->tab_data);
        }
        *hp = NULL;                                                          490
        return(0);
}

/*
```

```
 * Return the options for a particular filesystem.
 */
char *filsys_options(name, type, flags)
        char                    *name;
        int                     type;
        struct flags            *flags;                                 500
{
        register struct  _tab   *fp;
        int                     retval;
        char                    errbuf[100];
        regex_t                 reg;


        for (fp = optionsdb.tab; fp->regexp; fp++) {
                if (fp->explicit && ((flags->explicit && fp->explicit == -1) ||
                                        (!flags->explicit && fp->explicit == 1)))
                                                                        510
                        continue;
                if (!(fp->fs_type & type))
                        continue;


                if (retval=regcomp(&reg, fp->regexp, 0)) {
                        regerror(retval, &reg, errbuf, sizeof(errbuf));
                        fprintf(stderr, "Mountpoint config:  %s:  %s",
                                errbuf, fp->regexp);
                        config_abort();                                 520
                }
                if (regexec(&reg, name, 0, NULL, 0) == 0)
                        return(fp->tab_data);

        }
        return(NULL);
}
```

## A.3.8 detach.c

```
/*      Created by:     Robert French
 *
 *      $Source: /afs/rel-eng.athena.mit.edu/project/release/current/source/athena/athena.bin/attach/RCS/detach
 *      $Author: probe $
 *
 *      Copyright (c) 1988 by the Massachusetts Institute of Technology.
 */

static char *rcsid_detach_c = "$Header:  /afs/rel-eng.athena.mit.edu/project/release/current/source/at
                                                                        10
#include "attach.h"
#include <string.h>

/*
 * Detach the filesystem called name.
 */
```

```
detach(name, flags)
    char *name;
    struct flags *flags;                                                    20
{
    struct attach_list list = ATTACH_LIST_INITIALIZER;
    struct _attachtab *atp, at;
    struct fslock fslock;
    int status, bymntpt;
#ifdef ZEPHYR
    char instbfr[BUFSIZ];
#endif

    lock_attachtab(&list, flags);                                           30
    get_attachtab(&list, flags);
    bymntpt = 0;
    if (!(atp = attachtab_lookup(name, &list, flags))) {
        if (!(atp = attachtab_lookup_mntpt(name, &list, flags))) {
            unlock_attachtab(&list, flags);
            free_attachtab(&list, flags);
            fprintf(stderr, "%s:  Filesystem \"%s\" is not attached\n",
                    config.progname, name);
            flags->status = ERR_DETACHNOTATTACHED;
            return (FAILURE);                                               40
        } else
                bymntpt = 1;
    }

    switch (atp->status) {
    case STATUS_ATTACHING:
        if (!flags->force && really_in_use(name, flags)) {
            fprintf(stderr, "%s:  Filesystem \"%s\" is being attached by another process\n",
                    config.progname, name);
            flags->status = ERR_DETACHINUSE;                                50
            unlock_attachtab(&list, flags);
            free_attachtab(&list, flags);
            return (FAILURE);
        }
        /*
         * Derelict entry...here might be a good place for attachtab/mtab
         * reconciliation.
         */
        break;
    case STATUS_DETACHING:                                                  60
        if (!flags->force && really_in_use(name, flags)) {
            fprintf(stderr, "%s:  Filesystem \"%s\" already being detached by another process\n",
                    config.progname, name);
            flags->status = ERR_DETACHINUSE;
            unlock_attachtab(&list, flags);
            free_attachtab(&list, flags);
            return (FAILURE);
        }
        break;
    }                                                                       70
```

```
/* Note: attachtab still locked at this point */

start_critical_code();
if (flags->clean_detach) {
        if (clean_attachtab(atp, flags)) {
                put_attachtab(&list, flags);
                unlock_attachtab(&list, flags);
                free_attachtab(&list, flags);
                end_critical_code();                                    80
                return(SUCCESS);
        }
        /*
         * If we fall through, it means we should detach the filesystem
         */
}
if (atp->status == STATUS_ATTACHED) {
        if (flags->override) {          /* Override _all_ owners and detach */
                atp->nowners = 0;
                atp->flags &= ~FLAG_LOCKED;                             90
                status = SUCCESS;
        } else if (config.owner_check && !flags->clean_detach) {
                if (del_an_owner(atp, flags->owner_uid)) {
                        int ret = SUCCESS;
                        fprintf(stderr, "%s:  Filesystem \"%s\" wanted by others, not unmounted.\n",
                                config.progname, name);
                        flags->status = ERR_ATTACHINUSE;
                        put_attachtab(&list, flags);
                        unlock_attachtab(&list, flags);
                        if (atp->fs->type == TYPE_NFS && flags->do_nfsid)       100
                                ret = nfsid(atp->host, atp->hostaddr[0],
                                        MOUNTPROC_KUIDUMAP, 1,
                                        atp->hesiodname, 0,
                                        config.real_uid, flags);
                        free_attachtab(&list, flags);
                        end_critical_code();
                        return(ret);
                }
                status = SUCCESS;
        }                                                               110
        if (!flags->override && atp->flags & FLAG_LOCKED) {
                flags->status = ERR_DETACHNOTALLOWED;
                if (trusted_user(config.real_uid))
                  fprintf(stderr,
                        "%s:  Filesystem \"%s\" locked, use -override to detach it.\n",
                        config.progname, name);
                else
                  fprintf(stderr, "%s:  Filesystem \"%s\" locked, can't detach\n",
                        config.progname, name);
                put_attachtab(&list, flags);                            120
                unlock_attachtab(&list, flags);
                free_attachtab(&list, flags);
                end_critical_code();
                return(FAILURE);
        }
```

```
    }
    atp->status = STATUS_DETACHING;
    put_attachtab(&list, flags);
    mark_in_use(name, &fslock, flags);                                    130
    at = *atp;
    unlock_attachtab(&list, flags);
    free_attachtab(&list, flags);

    if (at.fs->detach)
            status = (at.fs->detach)(&at, &list, flags);
    else {
            fprintf(stderr, "%s:  Can't detach filesystem type \"%s\"\n",
                    config.progname, at.fs->name);
            flags->status = ERR_FATAL;                                    140
            return(FAILURE);
    }

    if (status == SUCCESS) {
        if (flags->verbose)
                printf("%s:  %s detached\n", config.progname, at.hesiodname);
        if (at.fs->flags & AT_FS_MNTPT)
                rm_mntpt(&at, flags);
        lock_attachtab(&list, flags);
        get_attachtab(&list, flags);                                      150
        if (bymntpt)
            attachtab_delete(attachtab_lookup_mntpt(at.mntpt, &list, flags),
                             &list, flags);
        else
            attachtab_delete(attachtab_lookup(at.hesiodname, &list, flags),
                             &list, flags);
        put_attachtab(&list, flags);
        mark_in_use(NULL, &fslock, flags);
        unlock_attachtab(&list, flags);
        /*                                                                160
         * Do Zephyr stuff as necessary
         */
#ifdef ZEPHYR
        if (flags->use_zephyr && at.fs->flags & AT_FS_REMOTE) {
                sprintf(instbfr, "%s:%s", at.host, at.hostdir);
                zephyr_addsub(instbfr, flags);
                if (!host_occurs(at.host, &list, flags))
                        zephyr_addsub(at.host, flags);
      }
#endif                                                                    170
        free_attachtab(&list, flags);
    } else {
        lock_attachtab(&list, flags);
        get_attachtab(&list, flags);
        at.status = STATUS_ATTACHED;
        attachtab_replace(&at, &list, flags);
        put_attachtab(&list, flags);
        mark_in_use(NULL, &fslock, flags);
        unlock_attachtab(&list, flags);
```

```
        free_attachtab(&list, flags);                                    180
    }
    end_critical_code();
    return (status);
}

/*
 * Detach all filesystems.  Read through the attachtab and call detach
 * on each one.
 */
                                                                         190
void detach_all(flags)
    struct flags *flags;
{
    struct attach_list          list = ATTACH_LIST_INITIALIZER;
    struct attach_list          inner_list = ATTACH_LIST_INITIALIZER;
    struct _attachtab           *atp, *next;
    int                         tempexp;
    extern struct _attachtab    *attachtab_last, *attachtab_first;
    struct _fstypes             *fs_type = NULL;
                                                                         200
    lock_attachtab(&list, flags);
    get_attachtab(&list, flags);
    unlock_attachtab(&list, flags);

    if (flags->filsys_type)
            fs_type = get_fs(flags->filsys_type);
    tempexp = flags->explicit;
    atp = list.attachtab_last;
    while (atp) {
            next = atp->prev;                                            210
            flags->explicit = atp->explicit;
            if ((flags->override || !config.owner_check || flags->clean_detach
                    || is_an_owner(atp,flags->owner_uid)) &&
                (!flags->filsys_type || fs_type == atp->fs)) {
                    if (atp->flags & FLAG_LOCKED) {
                            del_an_owner(atp, flags->owner_uid);
                            lock_attachtab(&inner_list, flags);
                            get_attachtab(&inner_list, flags);
                            attachtab_replace(atp, &inner_list, flags);
                            put_attachtab(&inner_list, flags);           220
                            unlock_attachtab(&inner_list, flags);
                            fprintf(stderr,
                                    "%s:  Filesystem \"%s\" locked, not unmounted.\n",
                                    config.progname, atp->hesiodname);
                    } else
                            detach(atp->hesiodname, flags);
            }
            atp = next;
    }
    free_attachtab(&inner_list, flags);                                  230
    free_attachtab(&list, flags);
    flags->explicit = tempexp;
}
```

```
/*
 * Detach routine for the NULL filesystem.  This is for cleanup
 * purposes only.
 */

null_detach(at, list, flags)                                          240
    struct _attachtab *at;
    struct attach_list *list;
    struct flags *flags;
{
    if (flags->debug_flag)
        printf("Detaching null filesystem %s...\n", at->hesiodname);

    return(SUCCESS);
}
                                                                      250

/*
 * Detach all filesystems from a specified host.  Read through the
 * attachtab and call detach on each one.
 */

void detach_host(host, flags)
    char *host;
    struct flags *flags;
{                                                                     260
    struct attach_list      list = ATTACH_LIST_INITIALIZER;
    struct attach_list      inner_list = ATTACH_LIST_INITIALIZER;
    struct _attachtab       *atp, *next;
    int tempexp;
    extern struct _attachtab        *attachtab_last, *attachtab_first;
    struct _fstypes         *fs_type = NULL;
    register int i;

    lock_attachtab(&list, flags);
    get_attachtab(&list, flags);                                      270
    unlock_attachtab(&list, flags);

    if (flags->filsys_type)
        fs_type = get_fs(flags->filsys_type);
    tempexp = flags->explicit;
    atp = list.attachtab_last;
    while (atp) {
        next = atp->prev;
        flags->explicit = atp->explicit;
        if ((flags->override || !config.owner_check || flags->clean_detach  280
            || is_an_owner(atp, flags->owner_uid)) &&
           (!flags->filsys_type || fs_type == atp->fs)) {
           for (i=0; i<MAXHOSTS && atp->hostaddr[i].s_addr; i++)
                if (host_compare(host, (char *) inet_ntoa(atp->hostaddr[i]),
                                 flags)) {
                    if (atp->flags & FLAG_LOCKED) {
                        del_an_owner(atp, flags->owner_uid);
```

```
                        lock_attachtab(&inner_list, flags);
                        get_attachtab(&inner_list, flags);
                        attachtab_replace(atp, &inner_list, flags);
                        put_attachtab(&inner_list, flags);
                        unlock_attachtab(&inner_list, flags);
                        fprintf(stderr,
                                "%s:  Filesystem \"%s\" locked, not unmounted.\n",
                                config.progname, atp->hesiodname);
                    } else
                        detach(atp->hesiodname, flags);
                    break;
                }
        }
        free(atp);
        atp = next;
    }
    free_attachtab(&inner_list, flags);
    free_attachtab(&list, flags);
    flags->explicit = tempexp;
}
```

## A.3.9 emul_re.c

```
/*
 * This is for A/UX.  It is a wrapper around the C library regex functions.
 *
 * $Id: emul_re.c,v 1.2 91/07/06 14:56:34 probe Exp $
 */

#ifdef _AUX_SOURCE

static char *re;
int Error;

char *re_comp(s)
    char *s;
{
    if (!s)
        return 0;
    if (re)
        free(re);

    if (!(re = regcmp(s, (char *)0)))
        return "Bad argument to re_comp";

    return 0;
}

int re_exec(s)
    char *s;
{
    return regex(re, s) != 0;
```

```
}

#endif
```

## A.3.10 getrealm.c

```
/*
 * $Source: /afs/rel-eng.athena.mit.edu/project/release/current/source/athena/athena.bin/attach/RCS/getre
 * $Author: probe $
 *
 * Copyright 1988 by the Massachusetts Institute of Technology.
 *
 * For copying and distribution information, please see the file
 * <mit-copyright.h>.
 *
 * routine to convert hostname into realm name.
 */

#include "config.h"
#ifdef OLD_KERBEROS

static char *rcsid_getrealm_c =
 "$Header:  /afs/rel-eng.athena.mit.edu/project/release/current/source/athena/athena.bin/at

#include <mit-copyright.h>
#include <strings.h>
#include <stdio.h>
#include <krb.h>
#include <sys/param.h>

/* for Ultrix and friends ... */
#ifndef MAXHOSTNAMELEN
#define MAXHOSTNAMELEN 64
#endif

/*
 * krb_realmofhost.
 * Given a fully-qualified domain-style primary host name,
 * return the name of the Kerberos realm for the host.
 * If the hostname contains no discernable domain, or an error occurs,
 * return the local realm name, as supplied by get_krbrlm()
 *
 * The format of each line of the translation file is:
 * domain_name kerberos_realm
 * -or-
 * host_name kerberos_realm
 *
 * domain_name should be of the form .XXX.YYY (e.g. .LCS.MIT.EDU)
 * host names should be in the usual form (e.g. FOO.BAR.BAZ)
 */

static char ret_realm[REALM_SZ+1];
```

```
char *
krb_realmofhost(host)
char *host;                                                          50
{
        char *domain, *trans_idx;
        FILE *trans_file;
        char trans_host[MAXHOSTNAMELEN+1];
        char trans_realm[REALM_SZ+1];
        int retval;

        domain = index(host, '.');

        /* prepare default */                                        60
        if (domain) {
                strncpy(ret_realm, &domain[1], REALM_SZ);
                ret_realm[REALM_SZ] = '\0';
        } else {
                get_krbrlm(ret_realm, 1);
        }

        if ((trans_file = fopen(KRB_RLM_TRANS, "r")) == (FILE *) 0) {
                /* krb_errno = KRB_NO_TRANS */
                return(ret_realm);                                   70
        }
        while (1) {
                if ((retval = fscanf(trans_file, "%s %s",
                                        trans_host, trans_realm)) != 2) {
                        if (retval == EOF) {
                                fclose(trans_file);
                                return(ret_realm);
                        }
                        continue;       /* ignore broken lines */
                }                                                    80
                trans_host[MAXHOSTNAMELEN] = '\0';
                trans_realm[REALM_SZ] = '\0';
                if (!strcasecmp(trans_host, host)) {
                        /* exact match of hostname, so return the realm */
                        (void) strcpy(ret_realm, trans_realm);
                        fclose(trans_file);
                        return(ret_realm);
                }
                if ((trans_host[0] = '.') && domain) {
                        /* this is a domain match */                 90
                        if (!strcasecmp(trans_host, domain)) {
                                /* domain match, save for later */
                                (void) strcpy(ret_realm, trans_realm);
                                continue;
                        }
                }
        }
}
#endif
```

## A.3.11 main.c

```
/*
 * $Id: main.c,v 1.32 94/03/25 15:54:51 miki Exp $
 *
 * Copyright (c) 1988,1992 by the Massachusetts Institute of Technology.
 */

static char *rcsid_main_c = "$Id:  main.c,v 1.32 94/03/25 15:54:51 miki Exp $";

#include "attach.h"
#include <signal.h>                                                    10
#include <string.h>

struct _fstypes fstypes[] = {
   { "---", 0, -1, 0, NULL, NULL, null_detach, NULL },      /* The null type */
#ifdef NFS
   { "NFS", TYPE_NFS, MOUNT_NFS,
         AT_FS_MNTPT | AT_FS_REMOTE | AT_FS_MNTPT_CANON,
         "rwnm", nfs_attach, nfs_detach, nfs_explicit },
#endif
#ifdef RVD                                                             20
   { "RVD", TYPE_RVD, MOUNT_UFS,
         AT_FS_MNTPT | AT_FS_REMOTE | AT_FS_MNTPT_CANON,
         "rw", rvd_attach, rvd_detach, rvd_explicit },
#endif
#ifdef UFS
   { "UFS", TYPE_UFS, MOUNT_UFS,
         AT_FS_MNTPT | AT_FS_MNTPT_CANON,
         "rw", ufs_attach, ufs_detach, ufs_explicit },
#endif
#ifdef AFS                                                             30
   { "AFS", TYPE_AFS, -1,
         AT_FS_MNTPT | AT_FS_PARENTMNTPT,
         "nrw", afs_attach, afs_detach, afs_explicit },
#endif
   { "ERR", TYPE_ERR, -1, 0, NULL, err_attach, NULL, NULL },
   { "MUL", TYPE_MUL, -1, 0, "-", mul_attach, mul_detach, NULL },
   { NULL, -1, 0, 0, NULL, NULL, NULL, NULL }
};

struct config config;                                                  40

static int fsidcmd _P((int argc, char **argv, struct flags *flags));
static void fsid_all _P((int op, char *ops, struct attach_list *list,
                          struct flags *flags));
static int fsid_filsys _P((struct _attachtab *atp, int op, char *ops,
                           char *filsys, int uid, struct attach_list *list,
                           struct flags *flags));
static int attachcmd _P((int argc, char **argv, struct flags *flags));
static void finish_thread _P((pthread_t thread, int *error_status_ptr));
static int detachcmd _P((int argc, char **argv, struct flags *flags));  50
static int zinitcmd _P((int argc, char **argv, struct flags *flags));
```

```
/*
 * Primary entry point -- look at argv[0] to figure out who we are and
 * dispatch to the proper handler accordingly.
 */

main(argc, argv)
    int argc;
    char *argv[];                                                    60
{
    char            *ptr;
    struct flags flags;
#ifdef POSIX
    struct sigaction sig;
#endif

    hes_init();

    /* Stop overriding my explicit file modes! */                    70
    umask(022);

    /* Install signal handlers */
#ifdef POSIX
    sig.sa_handler = sig_trap;
    sigemptyset(&sig.sa_mask);
    sig.sa_flags = 0;
    sigaction(SIGTERM, &sig, NULL);
    sigaction(SIGINT, &sig, NULL);
    sigaction(SIGHUP, &sig, NULL);                                   80
#else
    signal(SIGTERM, sig_trap);
    signal(SIGINT, sig_trap);
    signal(SIGHUP, sig_trap);
#endif

    config.verbose = 1;
    config.debug_flag = 0;
    config.owner_check = 0;
    config.owner_list = 1;                                           90
    config.exp_allow = 1;
    config.exp_mntpt = 1;
    config.real_uid = getuid();
    config.effective_uid = geteuid();
    config.abort_msg = "Operation aborted\n";

    config.progname = argv[0];
    ptr = strrchr(config.progname, '/');
    if (ptr)
        config.progname = ptr+1;                                     100

    config.attachtab_fn = strdup(ATTACHTAB);
    config.mtab_fn = strdup(MTAB);
#ifdef AFS
    config.aklog_fn = strdup(AKLOG_FULLNAME);
    config.afs_mount_dir = strdup(AFS_MOUNT_DIR);
```

```
#endif
#ifdef NFS
    config.nfs_mount_dir = strdup("");
#endif                                                              110
    config.fsck_fn = strdup(FSCK_FULLNAME);

    /* Initialize flags common to several functions. */
    flags.map_anyway = 0;
    flags.do_nfsid = 1;
    flags.print_path = 0;
    flags.explicit = 0;
    flags.override = 0;
    flags.clean_detach = 0;
    flags.force = 0;                                                120
    flags.lock_filesystem = 0;
    flags.clean_detach = 0;
#ifdef USE_ZEPHYR
    flags.use_zephyr = 1;
#endif
    flags.filsys_type = NULL;
    flags.spoofhost = NULL;
    flags.owner_uid = config.real_uid;
    flags.skip_fsck = 0;
    flags.status = ERR_NONE;                                        130
    flags.output = NULL;

    /*
     * User can explicitly specify attach, detach, or nfsid by using the
     * -P option as the first command option.
     */

    if (argv[1] && !strncmp(argv[1], "-P", 2)) {
            config.progname = argv[1]+2;
            if (*config.progname) {                                 140
                    argv += 1;
                    argc -= 1;
            } else {
                    if (argv[2]) {
                            config.progname = argv[2];
                            argv += 2;
                            argc -= 2;
                    } else {
                            fprintf(stderr,
                                    "Must specify attach, detach nfsid, fsid, or zinit!\n");
                            exit(ERR_BADARGS);
                    }
            }
    }

    if (!strcmp(config.progname, ATTACH_CMD))
            return attachcmd(argc, argv, &flags);
    if (!strcmp(config.progname, DETACH_CMD))
            return detachcmd(argc, argv, &flags);
#ifdef KERBEROS                                                     160
```

```
#ifdef NFS
    if (!strcmp(config.progname, NFSID_CMD))
    {
        flags.filsys_type = "NFS";
        return fsidcmd(argc, argv, &flags);
    }
#endif
    if (!strcmp(config.progname, FSID_CMD))
        return fsidcmd(argc, argv, &flags);
#endif                                                          170
#ifdef ZEPHYR
    if (!strcmp(config.progname, ZINIT_CMD))
        return zinitcmd(argc, argv, &flags);
#endif

    fprintf(stderr, "Not invoked with attach, detach, nfsid, fsid, or zinit!\n");
    exit(ERR_BADARGS);
}

#ifdef KERBEROS                                                 180
/*
 * Command handler for nfsid.
 */
static int fsidcmd(argc, argv, flags)
    int argc;
    char **argv;
    struct flags *flags;
{
    int gotname, i, op, filsysp, hostp;
#ifdef AFS                                                      190
    int    cell_sw;
#endif
    char *ops;
    struct hostent *hent;
    struct hostent_answer answer;
    char hostname[BUFSIZ];
    struct attach_list list = ATTACH_LIST_INITIALIZER;
    struct _attachtab *atp;
    struct in_addr addr;
    static struct command_list options[] = {                    200
        { "-verbose", "-v" },
        { "-quiet", "-q" },
        { "-debug", "-d" },
        { "-map", "-m" },
        { "-unmap", "-u" },
        { "-purge", "-p" },
        { "-purgeuser", "-r" },
        { "-spoofhost", "-s" },
        { "-filsys", "-f" },
        { "-host", "-h" },                                      210
#ifdef AFS
        { "-cell", "-c" },
#endif
        { "-user", "-U" },
```

```
        { "-all", "-a" },
        { 0, 0 }};

    check_root_privs(flags);
    read_config_file(ATTACHCONFFILE);

    gotname = 0;
    filsysp = 1;
    hostp = 0;
    flags->verbose = config.verbose;
    flags->debug_flag = config.debug_flag;
    flags->status = ERR_NONE;
#ifdef AFS
    cell_sw = 0;
#endif

    op = MOUNTPROC_KUIDMAP;
    ops = "mapped";

#ifdef ATHENA_COMPAT73
    if (flags->filsys_type && !strcmp(flags->filsys_type, "NFS")) {
        hostp = 1;
        filsysp = 0;
    } else {
        hostp = filsysp = 1;
    }
#endif

    for (i=1;i<argc;i++) {
        if (*argv[i] == '-') {
            switch (internal_getopt(argv[i], options)) {
            case 'v':
                flags->verbose = 1;
                break;
            case 'q':
                flags->verbose = 0;
                break;
            case 'd':
                config.debug_flag = 1;
                break;
            case 'm':
                op = MOUNTPROC_KUIDMAP;
                ops = "mapped";
                break;
            case 'u':
                op = MOUNTPROC_KUIDUMAP;
                ops = "unmapped";
                break;
            case 'r':
                op = MOUNTPROC_KUIDUPURGE;
                ops = "mappings user-purged";
                break;
            case 'p':
                if (!trusted_user(config.real_uid)) {
```

```
                fprintf(stderr,
                        "%s:  nfsid purge is a privileged operation\n",
                        config.progname);
                return ERR_NFSIDPERM;
            }
            op = MOUNTPROC_KUIDPURGE;
            ops = "mappings purged";
            break;
#ifdef AFS
        case 'c':
            filsysp = 0;
            cell_sw = 1;
            break;
#endif
        case 'f':
            filsysp = 1;
            hostp = 0;
            break;
        case 'h':
            filsysp = 0;
            hostp = 1;
            break;
        case 'U':
            ++i;
            if (trusted_user(config.real_uid)) {
                if (!argv[i]) {
                    fprintf(stderr, "%s:  Username required with -U.\n",
                            config.progname);
                    return ERR_BADARGS;
                }
                flags->owner_uid = parse_username(argv[i]);
            } else {
                fprintf(stderr,
            "%s:  You are not authorized to use the -user option\n",
                        config.progname);
            }
            break;
        case 's':
            if (i == argc-1) {
                fprintf(stderr, "%s:  No spoof host specified\n",
                        config.progname);
                return (ERR_BADARGS);
            }
            flags->spoofhost = argv[++i];
            break;
        case 'a':
            fsid_all(op, ops, &list, flags);
            gotname = 2;
            break;
        default:
            fprintf(stderr, "%s:  Unknown switch %s\n",
                    config.progname, argv[i]);
            return (ERR_BADARGS);
        }
```

```
            continue;
        }
        gotname++;

#ifdef AFS
        if (cell_sw) {
            afs_auth_to_cell(argv[i], flags);
            continue;                                                    330
        }
#endif

        if (filsysp) {
            lock_attachtab(&list, flags);
            get_attachtab(&list, flags);
            unlock_attachtab(&list, flags);
            /*
             * Lookup the specified filsys name and perform an nfsid
             * on the host associated with it.                           340
             */
            if (atp = attachtab_lookup(argv[i], &list, flags)) {
                if (atp->fs->type == TYPE_MUL)
                    gotname = 2;
                fsid_filsys(atp, op, ops, argv[i], flags->owner_uid, &list,
                        flags);
                continue;
            } else if (!hostp) {
                flags->status = ERR_NFSIDNOTATTACHED;
                fprintf(stderr, "%s:  %s not attached\n",                350
                        config.progname, argv[i]);
            }
        }

        if (hostp) {
            /*
             * Perform an nfsid on the specified host.
             */
            hent = gethostbyname_r(argv[i], &answer);
            if (!hent) {                                                 360
                fprintf(stderr, "%s:  Can't resolve %s\n", config.progname,
                        argv[i]);
                flags->status = ERR_NFSIDBADHOST;
            }
            else {
                strcpy(hostname, hent->h_name);
#ifdef POSIX
                memmove(&addr, hent->h_addr_list[0], 4);
#else
                bcopy(hent->h_addr_list[0], &addr, 4);                   370
#endif
                if ((nfsid(hostname, addr, op, 1, "nfsid", 0,
                        flags->owner_uid, flags) == SUCCESS) &&
                    flags->verbose)
                    printf("%s:  %s %s\n", config.progname, hostname, ops);
            }
```

```
        }
    }

    if (gotname == 1)                                                    380
        return (flags->status);
    if (gotname > 1)
        return (flags->status ? ERR_SOMETHING : ERR_NONE);

    fprintf(stderr, "Usage:  nfsid [options] [host host ...]  or [filsys filsys ...]\n");
    return (ERR_BADARGS);
}
#endif

static void fsid_all(op, ops, list, flags)                               390
    int op;
    char *ops;
    struct attach_list *list;
    struct flags *flags;
{
    struct _attachtab *atp;

    /*
     * Read the attachtab one entry at a time, and perform the
     * operation on each host found therein.  Note this                400
     * will even include hosts that are associated with
     * filesystems in the process of being attached or
     * detached.  Assume that the -a option implies more
     * than one file for the exit status computation.
     */
    lock_attachtab(list, flags);
    get_attachtab(list, flags);
    unlock_attachtab(list, flags);
    atp = list->attachtab_first;
    while (atp) {                                                        410
        if (atp->fs->type == TYPE_MUL) {
            /* Do nothing
             * Type MUL filesystems are authenticated
             * by their individual parts.
             */
        } else if ((op == MOUNTPROC_KUIDMAP ||
                    op == MOUNTPROC_KUIDUMAP) &&
                   !is_an_owner(atp, flags->owner_uid)) {
            /* Do nothing
             * Only map/unmap to filesystems that the                  420
             * user has attached.
             *
             * Purges apply to ALL attached filesystems.
             */
        } else if (atp->fs->type == TYPE_NFS) {
            if ((op == MOUNTPROC_KUIDMAP ||
                 op == MOUNTPROC_KUIDUMAP) &&
                atp->mode == 'n') {
                /* Do nothing */
            } else if (nfsid(atp->host, atp->hostaddr[0],                430
```

```
                        op, 1, atp->hesiodname, 0,
                        flags->owner_uid, flags) == SUCCESS &&
                    flags->verbose)
                printf("%s:  %s %s\n", config.progname,
                    atp->hesiodname, ops);
#ifdef AFS
        } else if (atp->fs->type == TYPE_AFS &&
                  op == MOUNTPROC_KUIDMAP) {
            /* We only support map operations on AFS */
            if (atp->mode != 'n' &&                                    440
                (afs_auth(atp->hesiodname, atp->hostdir, flags)
                 == SUCCESS) && flags->verbose)
                printf("%s:  %s %s\n", config.progname,
                    atp->hesiodname, ops);
#endif
        } else {
            if (flags->verbose)
                printf("%s:  %s ignored (operation not supported on %s)\n",
                    config.progname, atp->hesiodname,
                    atp->fs->name);                                    450
        }
        atp = atp->next;
    }
    free_attachtab(list, flags);
}

static int fsid_filsys(atp, op, ops, filsys, uid, list, flags)
    struct _attachtab *atp;
    int op;
    char *ops;                                                         460
    char *filsys;
    int uid;
    struct attach_list *list;
    struct flags *flags;
{
        char mul_buf[BUFSIZ], *cp;

        if (atp->fs->type == TYPE_MUL) {
                strcpy(mul_buf, atp->hostdir);
                cp = strtok(mul_buf, ",");                             470
                while (cp) {
                        atp = attachtab_lookup(cp, list, flags);
                        if (atp)
                                fsid_filsys(atp, op, ops, atp->hesiodname, uid,
                                        list, flags);
                        cp = strtok(NULL, ",");
                }
#ifdef NFS
        } else if (atp->fs->type == TYPE_NFS) {
                if ((nfsid(atp->host, atp->hostaddr[0], op, 1,         480
                          filsys, 0, flags->owner_uid, flags) == SUCCESS) &&
                    flags->verbose)
                        printf("%s:  %s %s\n", config.progname, filsys, ops);
#endif
```

```
#ifdef AFS
        } else if (atp->fs->type == TYPE_AFS) {
                if (op == MOUNTPROC_KUIDMAP &&
                    (afs_auth(atp->hesiodname, atp->hostdir, flags) == SUCCESS)
                    && flags->verbose)
                        printf("%s:   %s  %s\n", config.progname, filsys, ops);
#endif
        }
}

static int attachcmd(argc, argv, flags)
    int argc;
    char **argv;
    struct flags *flags;
{
    struct flags *flags_copy;
    int gotname, i, num_threads = 0;
    int print_host = 0;
    int print_banner = 1;
    int parallel = 0;
    int error_status = ERR_NONE;
    pthread_t threads[100];
    pthread_attr_t attr;

    static struct command_list options[] = {
        { "-verbose", "-v" },
        { "-quiet", "-q" },
        { "-force", "-f" },
        { "-printpath", "-p" },
        { "-lookup", "-l" },
        { "-debug", "-d" },
        { "-map", "-y" },
        { "-nomap", "-n" },
        { "-remap", "-g" },
        { "-noremap", "-a" },
#ifdef ZEPHYR
        { "-zephyr", "-z" },
        { "-nozephyr", "-h" },
#endif /* ZEPHYR */
        { "-readonly", "-r" },
        { "-write", "-w" },
        { "-mountpoint", "-m" },
        { "-noexplicit", "-x" },
        { "-explicit", "-e" },
        { "-type", "-t" },
        { "-mountoptions", "-o" },
        { "-spoofhost", "-s" },
        { "-nosetuid", "-N" },
        { "-setuid", "-S" },
        { "-nosuid", "-N" },
        { "-suid", "-S" },
        { "-override", "-O" },
        { "-skipfsck", "-F" },
        { "-lock", "-L" },
```

```
        { "-user", "-U" },
        { "-host", "-H" },
        { "-multithreaded", "-M" },
        { 0, 0 }};

pthread_attr_init(&attr);
pthread_attr_setschedpolicy(&attr, SCHED_FIFO);

read_config_file(ATTACHCONFFILE);

check_root_privs(flags);
config.default_suid = 0;

gotname = 0;

flags->verbose = config.verbose;
flags->debug_flag = config.debug_flag;
flags->explicit = 0;
flags->force = 0;
flags->lookup = 0;
flags->mntpt = (char *)NULL;
flags->override_mode = '\0';
flags->override_suid = -1;              /* -1 means use default */
flags->mount_options = "";
flags->map_anyway = 1;
flags->print_path = 0;

if (argc == 1)
        return (attach_print(0, 1, flags));

for (i=1;i<argc;i++) {
        if (*argv[i] == '-') {
                switch (internal_getopt(argv[i], options)) {
                case 'v':
                        flags->verbose = 1;
                        flags->print_path = 0;
                        break;
                case 'l':
                        flags->lookup = 1;
                        break;
                case 'q':
                        flags->verbose = 0;
                        break;
                case 'd':
                        flags->debug_flag = 1;
                        break;
                case 'y':
                        flags->do_nfsid = 1;
                        break;
                case 'n':
                        flags->do_nfsid = 0;
                        flags->map_anyway = 0;
                        break;
                case 'p':
```

```
                flags->print_path = 1;
                flags->verbose = 0;
                break;
        case 'm':
                if (i == argc - 1) {
                   fprintf(stderr, "%s:  No mount point specified\n",
                           config.progname);
                   return (ERR_BADARGS);                                    600
                }
                if (config.exp_mntpt || trusted_user(config.real_uid)) {
                        flags->mntpt = argv[++i];
                } else {
                        fprintf(stderr,
                "%s:  You are not allowed to use the -mountpoint option\n",
                                config.progname);
                        i++;
                }
                break;                                                      610
        case 'r':
                flags->override_mode = 'r';
                break;
        case 'w':
                flags->override_mode = 'w';
                break;
        case 'f':
                flags->force = 1;
                break;
        case 'g':                                                           620
                flags->map_anyway = 1;
                break;
        case 'a':
                flags->map_anyway = 0;
                break;
#ifdef ZEPHYR
        case 'z':
                flags->use_zephyr = 1;
                break;
        case 'h':                                                           630
                flags->use_zephyr = 0;
                break;
#endif /* ZEPHYR */
        case 'x':
                flags->explicit = 0;
                break;
        case 'e':
                if (config.exp_allow || trusted_user(config.real_uid))
                        flags->explicit = 1;
                else                                                        640
                        fprintf(stderr,
                "%s:  You are not allowed to use the -explicit option\n",
                                config.progname);
                break;
        case 't':
                if (i == argc-1) {
```

```
            fprintf(stderr, "%s:  No filesystem type specified\n",
                    config.progname);
            return (ERR_BADARGS);
        }
        flags->filsys_type = argv[++i];
        break;
    case 'o':
        if (i == argc-1) {
            fprintf(stderr, "%s:  No mount options specified\n",
                    config.progname);
            return (ERR_BADARGS);
        }
        flags->mount_options = argv[++i];
        break;
    case 's':
        if (i == argc-1) {
            fprintf(stderr, "%s:  No spoof host specified\n",
                    config.progname);
            return (ERR_BADARGS);
        }
        flags->spoofhost = argv[++i];
        break;
    case 'N':
        flags->override_suid = 0;
        break;
    case 'S':
        if (trusted_user(config.real_uid))
                flags->override_suid = 1;
        else {
                fprintf(stderr,

        "%s:  You are not authorized to the -setuid option\n",
                        config.progname);
        }
        break;
    case 'O':
        if (trusted_user(config.real_uid))
                flags->override = 1;
        else {
                fprintf(stderr,
        "%s:  You are not authorized to use -override option\n",
                        config.progname);
        }
        break;
    case 'L':
        if (trusted_user(config.real_uid))
                flags->lock_filesystem = 1;
        else {
                fprintf(stderr,
        "%s:  You are not authorized to use -lock option\n",
                        config.progname);
        }
        break;
    case 'F':
```

```
                flags->skip_fsck = 1;
                break;
        case 'U':
                ++i;
                if (trusted_user(config.real_uid)) {
                        if (argv[i])
                                flags->owner_uid = parse_username(argv[i]);
                } else {
                        fprintf(stderr,
                "You are not authorized to use the -user option\n",
                                config.progname);
                }
                break;
        case 'H':
                print_host++;
                break;
        case 'M':
                parallel = 1;
                break;
        default:
                fprintf(stderr, "%s:  Unknown switch %s\n", config.progname,
                        argv[i]);
                return (ERR_BADARGS);
        }
        continue;
    }
    gotname++;

    if (print_host) {
        attach_print(argv[i], print_banner, flags);
        print_banner = 0;
    } else {
        flags->fsname = argv[i];
        flags_copy = malloc(sizeof(struct flags));
        *flags_copy = *flags;
        pthread_create(&threads[num_threads], &attr, attach_threadfunc,
                        flags_copy);
        if (parallel)
                num_threads++;
        else
                finish_thread(threads[num_threads], &error_status);
    }

    flags->override_mode = '\0';
    flags->override_suid = -1;
    flags->override = 0;
    flags->lock_filesystem = 0;
    flags->mntpt = (char *)NULL;
}

/* Get thread return values and statuses. */
for (i = 0; i < num_threads; i++)
        finish_thread(threads[i], &error_status);
```

```
    /* Flush Zephyr subscriptions */
#ifdef ZEPHYR
    zephyr_sub(0, flags);
#endif /* ZEPHYR */

    if (gotname == 1)                                                    760
        return (error_status);
    if (gotname > 1)
        return (error_status ? ERR_SOMETHING : ERR_NONE);

    fprintf(stderr,
        "Usage: attach [options] filesystem [options] filesystem ...\n");
    return (ERR_BADARGS);
}

static void finish_thread(thread, error_status_ptr)                      770
    pthread_t thread;
    int *error_status_ptr;
{
    int status;
    void *thread_return;
    struct flags *flags;

    status = pthread_join(thread, &thread_return);
    if (status) {
        fprintf(stderr, "%s: pthread_join failed: %s\n", config.progname,  780
            strerror(errno));
        *error_status_ptr = status;
        exit(1);
    }
    flags = thread_return;
    if (flags->status != ERR_NONE)
        *error_status_ptr = flags->status;
    if (flags->output)
        fputs(flags->output, stdout);
}                                                                        790

static int detachcmd(argc, argv, flags)
    int argc;
    char **argv;
    struct flags *flags;
{
    int gotname, i;
    int    dohost;
    static struct command_list options[] = {
        { "-verbose", "-v" },                                            800
        { "-quiet", "-q" },
        { "-all", "-a" },
        { "-debug", "-d" },
        { "-unmap", "-y" },
        { "-nomap", "-n" },
#ifdef ZEPHYR
        { "-zephyr", "-z" },
        { "-nozephyr", "-h" },
```

```
#endif /* ZEPHYR */
        { "-type", "-t" },                                              810
        { "-explicit", "-e" },
        { "-noexplicit", "-x" },
        { "-force", "-f" },
        { "-spoofhost", "-s" },
        { "-override", "-O" },
        { "-host", "-H" },
        { "-user", "-U" },
        { "-clean", "-C" },
        { "-lint", "-L" },
        { 0, 0}};                                                       820

    check_root_privs(flags);
    read_config_file(ATTACHCONFFILE);

    gotname = 0;

    flags->verbose = config.verbose;
    flags->debug_flag = config.debug_flag;
    flags->explicit = 0;
    flags->force = 0;                                                   830
    flags->override = 0;
    flags->filsys_type = NULL;
    flags->status = ERR_NONE;
    dohost = 0;

    for (i=1;i<argc;i++) {
        if (*argv[i] == '-') {
            switch (internal_getopt(argv[i], options)) {
            case 'v':
                flags->verbose = 1;                                     840
                break;
            case 'q':
                flags->verbose = 0;
                break;
            case 'd':
                flags->debug_flag = 1;
                break;
            case 'y':
                flags->do_nfsid = 1;
                break;                                                  850
            case 'n':
                flags->do_nfsid = 0;
                break;
#ifdef ZEPHYR
            case 'z':
                flags->use_zephyr = 1;
                break;
            case 'h':
                flags->use_zephyr = 0;
                break;                                                  860
#endif /* ZEPHYR */
            case 'H':
```

```
        dohost = 1;
        break;
case 'f':
        flags->force = 1;
        break;
case 'x':
        flags->explicit = 0;
        break;                                                          870
case 'e':
        flags->explicit = 1;
        break;
case 't':
        if (i == argc-1) {
           fprintf(stderr, "%s:  No filesystem type specified\n",
                   config.progname);
           return (ERR_BADARGS);
        }
        flags->filsys_type = argv[++i];                                 880
        break;
case 'a':
        detach_all(flags);
        gotname = 2;
        break;
case 's':
        if (i == argc-1) {
           fprintf(stderr, "%s:  No spoof host specified\n",
                   config.progname);
           return (ERR_BADARGS);                                        890
        }
        flags->spoofhost = argv[++i];
        break;
case 'O':
        if (trusted_user(config.real_uid))
                flags->override = 1;
        else {
                fprintf(stderr,
        "%s:  You are not authorized to use -override option\n",
                        config.progname);                               900
        }
        break;
case 'U':
        ++i;
        if (trusted_user(config.real_uid)) {
                if (!argv[i]) {
                        fprintf(stderr, "%s:  Username required with -U.\n",
                                config.progname);
                        return (ERR_BADARGS);
                }                                                       910
                flags->owner_uid = parse_username(argv[i]);
        } else {
                fprintf(stderr,
        "%s:  You are not authorized to use the -user option\n",
                        config.progname);
        }
```

```
                break;
        case 'C':
                flags->clean_detach = 1;
                break;                                                    920
        case 'L':
                if (!trusted_user(config.real_uid)) {
                    fprintf(stderr,
                            "%s:  You are not authorized to use the -lint option\n",
                            config.progname);
                    return(ERR_BADARGS);
                } else {
                    lint_attachtab(flags);
                    gotname = 1;
                }                                                         930
                break;
        default:
                fprintf(stderr, "%s:  Unknown switch %s\n", config.progname,
                        argv[i]);
                return (ERR_BADARGS);
            }
            continue;
        }
        gotname++;
        if (dohost)                                                       940
                detach_host(argv[i], flags);
        else
                detach(argv[i], flags);
        dohost = 0;
    }

    /* Flush Zephyr unsubscriptions */
#ifdef ZEPHYR
    zephyr_unsub(0, flags);
#endif /* ZEPHYR */                                                       950

    if (gotname == 1)
            return (flags->status);
    if (gotname > 1)
            return (flags->status ? ERR_SOMETHING : ERR_NONE);

    fprintf(stderr,
            "Usage:  detach [options] filesystem [options] filesystem ...\n");
    return (ERR_BADARGS);
}                                                                         960

#ifdef ZEPHYR
static int zinitcmd(argc, argv, flags)
    int argc;
    char **argv;
    struct flags *flags;
{
    struct attach_list list = ATTACH_LIST_INITIALIZER;
    char instbfr[BUFSIZ];
    struct _attachtab *p;                                                 970
```

```
    int i;
    enum { USER_ONLY, ROOT_TOO, ALL_USERS } who = ROOT_TOO;

    static struct command_list options[] = {
            { "-verbose", "-v" },
            { "-quiet", "-q" },
            { "-all", "-a"},
            { "-me", "-m"},
            { "-debug", "-d" },
            { 0, 0}};                                                    980

    read_config_file(ATTACHCONFFILE);
    flags->verbose = config.verbose;
    flags->debug_flag = config.debug_flag;
    for (i=1;i<argc;i++) {
            if (*argv[i] == '-') {
                switch (internal_getopt(argv[i], options)) {
                  case 'v':
                        flags->verbose = 1;
                        break;                                           990
                  case 'q':
                        flags->verbose = 0;
                        break;
                  case 'd':
                        flags->debug_flag = 1;
                        break;
                  case 'm':
                        who = USER_ONLY;
                        break;
                  case 'a':                                              1000
                        who = ALL_USERS;
                        break;
                }
            }
    }

    lock_attachtab(&list, flags);
    get_attachtab(&list, flags);
    unlock_attachtab(&list, flags);
                                                                         1010
    for (p = list.attachtab_first; p; p = p->next ) {
            if (p->status == STATUS_ATTACHING)
                /*
                 * If it is being attached, someone else will
                 * deal with the zephyr subscriptions.
                 * (Also, all the information won't be here yet.)
                 */
                continue;
            if (who != ALL_USERS && !wants_to_subscribe(p, config.real_uid, who))
                continue;                                                1020
#ifdef AFS
            if (p->fs->type == TYPE_AFS)
                afs_zinit(p->hesiodname, p->hostdir, flags);
            else
```

```
#endif
            if (p->fs->flags & AT_FS_REMOTE) {
                sprintf(instbfr, "%s:%s", p->host, p->hostdir);
                zephyr_addsub(instbfr, flags);
                zephyr_addsub(p->host, flags);
            }                                                          1030
    }
    free_attachtab(&list, flags);
    return((zephyr_sub(1, flags) == FAILURE) ? flags->status : 0);
}
#endif /* ZEPHYR */
```

## A.3.12 mount.c

```
/*      Created by:         Robert French
 *
 *      $Id: mount.c,v 1.9 94/03/25 15:57:35 miki Exp $
 *
 *      Copyright (c) 1988 by the Massachusetts Institute of Technology.
 */

static char *rcsid_mount_c = "$Header:  /afs/dev.mit.edu/project/release/source/src/athena/bin/attach

#include "attach.h"                                                    10

#ifdef MOUNT_CMD

#ifdef _IBMR2
#include <sys/id.h>
#endif

mountfs(at, fsname, mopt, errorout, flags)
        struct  _attachtab *at;
        char    *fsname;                                               20
        struct  mntopts *mopt;
        int errorout;
        struct flags *flags;
{
        int status;

        if (setuid(0)) {
                fprintf(stderr, "Unable to change the uid to 0\n");
                return(FAILURE);
        }                                                              30

        switch (fork()) {
        case -1:
                fprintf(stderr, "Unable to fork\n");
                return(FAILURE);
                /* NOTREACHED */
        case 0:
                execl(MOUNT_CMD, MOUNT_CMD,
                    "-o", stropt(*mopt),
```

```
                fsname, at->mntpt,
                (char *)0);
            exit(1);
            /* NOTREACHED */
        default:
            wait(&status);
            break;
        }

        return(status ? FAILURE : SUCCESS);
}

#else /* !MOUNT_CMD */

#ifndef ultrix
#include <mntent.h>
#endif
#if defined(_AIX) && (AIXV < 30)
#include <sys/dstat.h>
#include <rpc/rpcmount.h>
#include <rpc/nfsmount.h>
struct ufs_args { char *fspec;};
#endif
#ifdef _AUX_SOURCE
#define mount(type,dir,flags,data)          fsmount(type,dir,flags,data)
#endif

/*
 * Mount a filesystem
 */
mountfs(at, fsname, mopt, errorout, flags)
        struct  _attachtab      *at;
        char    *fsname;
        struct  mntopts *mopt;
        int errorout;
        struct flags *flags;
{
        struct  mntent  mnt;
        union data {
#if defined(UFS) || defined(RVD)
            struct ufs_args    ufs_args;
#endif
#ifdef NFS
            struct nfs_args nfs_args;
#endif
        } data;

        mnt.mnt_fsname = fsname;
        mnt.mnt_dir = at->mntpt;
#if !defined(_AIX) && !defined(ultrix)
        mnt.mnt_type = (at->fs->mount_type==MOUNT_NFS) ? MNTTYPE_NFS
            : MNTTYPE_42;
#endif
```

```
        mnt.mnt_opts = stropt(*mopt);
        mnt.mnt_freq = 0;
#if defined(_AIX) && (AIXV<30)
        mnt.mnt_type = at->fs->mount_type == MOUNT_NFS ? "nfs" : "ufs";
        mnt.mnt_checkno = 0;
#else
        mnt.mnt_passno = 0;                                                    100
#endif
#if defined(sun)
        mnt.mnt_type = at->fs->mount_type == MOUNT_NFS ? "nfs" : "ufs";
#endif
        bzero(&data, sizeof(data));
        /* Already mounted? Why lose? */
        if (mounted(&mnt)) {
                fprintf(stderr,
                        "%s:  Already mounted, continuing...\n",
                        at->hesiodname);                                       110
                if (config.keep_mount)
                        at->flags |= FLAG_PERMANENT;
                return (SUCCESS);
        }
#if defined(UFS) || defined(RVD)
        if (at->fs->mount_type == MOUNT_UFS)
                if (mount_42(&mnt, &data.ufs_args) == FAILURE)
                        return (FAILURE);
#endif
#ifdef NFS                                                                     120
        if (at->fs->mount_type == MOUNT_NFS) {
                if (mount_nfs(at, mopt, &data.nfs_args, errorout,
                            flags) == FAILURE)
                        return (FAILURE);
#ifdef _AIX
                if (nfs_mount(mnt.mnt_dir, &data.nfs_args, mopt->flags) != 0) {
                        if (errorout)
                                fprintf(stderr,
                                        "%s:  Can't mount %s on %s - %s\n",
                                        at->hesiodname, fsname, mnt.mnt_dir,   130
                                        strerror(errno));
                        return (FAILURE);
                } else {
                  /* We need to get the filesystem's gfs for mtab */
                  struct dstat st_buf;
                  if(dstat(mnt.mnt_dir, &st_buf, sizeof(st_buf)) != 0) {
                    if (errorout)
                      fprintf(stderr,
                              "%s:  Can't stat %s to verify mount:  %s\n",
                              at->hesiodname, mnt.mnt_dir, strerror(errno));   140
                    return (FAILURE);
                  } else {
                    mnt.mnt_gfs = st_buf.dst_gfs;
                    goto done;
                  }
                }
#endif
```

```
        }
#endif /* NFS */
                                                                        150
#ifdef ultrix
        if (mount(mnt.mnt_fsname, mnt.mnt_dir, mopt->flags,
                  at->fs->mount_type, (char *)&data) < 0) {
#else /* !ultrix */
#ifdef _AIX
        if (mount(mnt.mnt_fsname, mnt.mnt_dir, mopt->flags) < 0) {
#else
#if defined(sun)
        if (mount(mnt.mnt_type, mnt.mnt_dir, M_NEWTYPE | mopt->flags, &data) < 0) {
#else                                                                   160
        if (mount(at->fs->mount_type, mnt.mnt_dir, mopt->flags, &data) < 0) {
#endif /* sun */
#endif
#endif /* ultrix */
                if (errorout) {
                        fprintf(stderr,
                                "%s:  Can't mount %s on %s - %s\n",
                                at->hesiodname, fsname, mnt.mnt_dir,
                                strerror(errno));
                        flags->status = ERR_SOMETHING;                  170
                }
                return (FAILURE);
        }

#ifdef _AIX
        else {
          struct dstat st_buf;
          if(dstat(mnt.mnt_dir, &st_buf, sizeof(st_buf)) != 0) {
            if (errorout)
              fprintf(stderr,                                           180
                      "%s:  Can't stat %s to verify mount:  %s\n",
                      at->hesiodname, mnt.mnt_dir, strerror(errno));
            return (FAILURE);
          } else {
            mnt.mnt_gfs = st_buf.dst_gfs;
          }
        }
#endif
#ifndef ultrix
    done:                                                               190
        addtomtab(&mnt);
#endif /* !ultrix */
        return (SUCCESS);
}

#if defined(UFS) || defined(RVD)
/*
 * Prepare to mount a 4.2 style file system
 */
                                                                        200
mount_42(mnt, args)
```

```
    struct mntent *mnt;
    struct ufs_args *args;
{
#ifndef ultrix
    args->fspec = mnt->mnt_fsname;
#endif
    return (SUCCESS);
}
                                                                    210
#endif /* UFS || RVD */

#ifdef NFS
/*
 * Mount an NFS file system
 */
mount_nfs(at, mopt, args, errorout, flags)
        struct  _attachtab      *at;
        struct  mntopts         *mopt;
        struct nfs_args *args;                                      220
        int errorout;
        struct flags *flags;
{
        static struct fhstatus fhs;
        static struct sockaddr_in sin;
        struct timeval timeout;
        CLIENT *client;
        enum clnt_stat rpc_stat;
        char    *hostdir = at->hostdir;
                                                                    230
        if (errored_out(at->hostaddr[0])) {
                if (errorout)
                        fprintf(stderr,
                "%s:  Ignoring %s due to previous host errors\n",
                                at->hesiodname, at->host);
                return (FAILURE);
        }

        if ((client = rpc_create(at->hostaddr[0], &sin, flags)) == NULL) {
                if (errorout)                                       240
                        fprintf(stderr, "%s:  Server %s not responding\n",
                                at->hesiodname, at->host);
                return (FAILURE);
        }

        client->cl_auth = spoofunix_create_default(flags->spoofhost,
                                                   config.real_uid);

        timeout.tv_usec = 0;
        timeout.tv_sec = 20;                                        250
        rpc_stat = clnt_call(client, MOUNTPROC_MNT, xdr_path, &hostdir,
                        xdr_fhstatus, &fhs, timeout);
        if (rpc_stat != RPC_SUCCESS) {
                mark_errored(at->hostaddr[0]);
                if (flags->debug_flag)
```

```
                clnt_perror(client, "RPC return status");
        if (!errorout)
                return(FAILURE);
        switch (rpc_stat) {
        case RPC_TIMEDOUT:                                              260
                fprintf(stderr,
                "%s:  Timeout while contacting mount daemon on %s\n",
                        at->hesiodname, at->host);
                break;
        case RPC_AUTHERROR:
                fprintf(stderr, "%s:  Authentication failed\n",
                        at->hesiodname, at->host);
                break;
        case RPC_PMAPFAILURE:
                fprintf(stderr, "%s:  Can't find mount daemon on %s\n",  270
                        at->hesiodname, at->host);
                break;
        case RPC_PROGUNAVAIL:
        case RPC_PROGNOTREGISTERED:
                fprintf(stderr,
                        "%s:  Mount daemon not available on %s\n",
                        at->hesiodname, at->host);
                break;
        default:
                fprintf(stderr,                                         280
                        "%s:  System error contacting server %s\n",
                        at->hesiodname, at->host);
                break;
        }
        return (FAILURE);
}

if (errno = fhs.fhs_status) {
        if (errorout) {
                if (errno == EACCES) {                                  290
                        fprintf(stderr,
                        "%s:  Mount access denied by server %s\n",
                                at->hesiodname, at->host);
                        flags->status = ERR_ATTACHNOTALLOWED;
                } else if (errno < sys_nerr) {
                        flags->status = (errno == ENOENT) ?
                          ERR_ATTACHNOFILSYS : ERR_ATTACHNOTALLOWED;
                        fprintf(stderr,
                "%s:  Error message returned from server %s:  %s\n",
                                at->hesiodname, at->host,               300
                                strerror(errno));
                } else {
                        flags->status = ERR_ATTACHNOTALLOWED;
                        fprintf(stderr,
                "%s:  Error status %d returned from server %s\n",
                                at->hesiodname, errno, at->host);
                }
        }
        return (FAILURE);
```

```
        }                                                                    310

        *args = mopt->tsa.nfs;
        args->hostname = at->host;
        args->fh = &fhs.fhs_fh;
        args->flags |= NFSMNT_HOSTNAME;
        if (mopt->nfs_port)
                sin.sin_port = mopt->nfs_port;
        else
                sin.sin_port = htons(NFS_PORT);   /* XXX should use portmapper */
        args->addr = &sin;                                                   320
#ifdef ultrix
        args->optstr = stropt(*mopt);
#endif
        return (SUCCESS);
}
#endif

#ifdef ultrix
mounted(mntck)
   struct mntent *mntck;                                                     330
{
   int done = 0;
   int fsloc = 0;
   int found = 0;
   struct fs_data fsdata;

   while (getmountent(&fsloc, &fsdata, 1) > 0) {
        if (!strcmp(fsdata.fd_path, mntck->mnt_dir) &&
           !strcmp(fsdata.fd_devname, mntck->mnt_fsname)) {
           return(1);                                                        340
           break;
        }
   }
   return(0);
}

#else /* !ultrix */

/*
 * Add an entry to /etc/mtab                                                350
 */

addtomtab(mnt)
   struct mntent *mnt;
{
   FILE *mnted;

   lock_mtab();
   mnted = setmntent(mtab_fn, "r+");
   if (!mnted || addmntent(mnted, mnt)) {                                    360
           fprintf(stderr, "Can't append to %s:  %s\n", mtab_fn,
                   strerror(errno));
           unlock_mtab();
```

```
            exit(ERR_FATAL);
    }
    endmntent(mnted);
    unlock_mtab();
}

mounted(mntck)                                                          370
    struct mntent *mntck;
{
    int found = 0;
    struct mntent *mnt;
    FILE *mnttab;

    lock_mtab();
    mnttab = setmntent(mtab_fn, "r");
    if (!mnttab) {
            fprintf(stderr, "Can't open %s for read\n", mtab_fn);       380
            exit(ERR_FATAL);
    }
    while (mnt = getmntent(mnttab)) {
        if (!strcmp(mnt->mnt_type, MNTTYPE_IGNORE))
            continue;
        if ((!strcmp(mntck->mnt_dir, mnt->mnt_dir)) &&
            (!strcmp(mntck->mnt_type, mnt->mnt_type))) {
                if (!strcmp(mnt->mnt_type, MNTTYPE_NFS)) {
                        if (nfs_fsname_compare(mntck->mnt_fsname,
                                                mnt->mnt_fsname)) {     390
                                found = 1;
                                break;
                        }
                } else if (!strcmp(mntck->mnt_fsname,
                                    mnt->mnt_fsname)) {
                        found = 1;
                        break;
                }
        }
    }                                                                   400
    endmntent(mnttab);
    unlock_mtab();
    return (found);
}

/*
 * Returns true if two NFS fsnames are the same.
 */
nfs_fsname_compare(fsname1, fsname2)
        char    *fsname1;                                               410
        char    *fsname2;
{
        char    host1[BUFSIZ], host2[BUFSIZ];
        char    *rmdir1, *rmdir2;

        (void) strcpy(host1, fsname1);
        (void) strcpy(host2, fsname2);
```

```
        if (rmdir1 = index(host1, ':'))
                *rmdir1++ = '\0';
        if (rmdir2 = index(host2, ':'))                                    420
                *rmdir2++ = '\0';
        if (host_compare(host1, host2)) {
                if (!rmdir1 || !rmdir2)
                        return(0);
                return(!strcmp(rmdir1, rmdir2));
        } else
                return(0);
}

#endif /* !ultrix */                                                       430

#endif /* !MOUNT_CMD */
```

## A.3.13 mul.c

```
/*
 * $Header: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/mul.c,v 1.5 94/03/25 15:58:31 mik
 *
 * Copyright (c) 1990 by the Massachusetts Institute of Technology.
 */

static char *rcsid_mul_c = "$Header:  /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/R


#include "attach.h"                                                        10
#include <string.h>

int mul_attach(atp, mopt, errorout, flags)
struct _attachtab *atp;
struct mntopts *mopt;
int errorout;
struct flags *flags;
{
        char mul_buf[BUFSIZ], *cp = mul_buf, *mp;
                                                                           20
        strcpy(mul_buf, atp->hostdir);
        while (mp = cp) {
                cp = strchr(mp, ',');
                if (cp)
                        *cp = '\0';
                attach(mp, flags);
                if (cp)
                        cp++;
        }
        return SUCCESS;                                                    30
}

int mul_detach(atp, list, flags)
struct _attachtab *atp;
struct attach_list *list;
```

```
struct flags *flags;
{
    int status = SUCCESS;
    char mul_buf[BUFSIZ], *cp;
                                                                        40
    strcpy(mul_buf, atp->hostdir);
    cp = &mul_buf[strlen(mul_buf)];
    while (cp-- >= mul_buf) {
        flags->explicit = 0;
        if (cp < mul_buf || *cp == ',') {
            if (attachtab_lookup(cp+1, list, flags)
                && detach(cp+1, flags) != SUCCESS
                && flags->status != ERR_DETACHNOTATTACHED)
                status = FAILURE;
            if (cp >= mul_buf)                                          50
                *cp = '\0';
        }
    }

    flags->status = (status == SUCCESS) ? ERR_NONE : ERR_SOMETHING;
    return status;
}
```

## A.3.14 nfs.c

```
/*
 *      $Source: /afs/dev.mit.edu/source/src.77/athena/bin/attach/RCS/nfs.c,v $
 *      $Author: cfields $
 *
 *      Copyright (c) 1988 by the Massachusetts Institute of Technology.
 */

static char *rcsid_nfs_c = "$Header:  /afs/dev.mit.edu/source/src.77/athena/bin/attach/RCS/nfs.c

#include "attach.h"                                                     10
#ifdef NFS

/* If the timeout is not explicitly specified by the attach command,
 * it is set to 20 tenths of a second (i.e. 2 seconds).  This should
 * really be set in the kernel, but due to release shakedown timing,
 * it's kludged in here.
 * Similarly with retransmissions.
 *
 * The kernel will double the timeout on each retry until it reaches a
 * max of 60 seconds, at which point it uses 60 seconds for the timeout.  20
 *
 * current kernel defaults: 4 retrans, 7 tenths sec timeout.
 * -JTK, 24 Oct 88
 *
 * The new values are: 6 retrans, 8 tenths sec timeout.
 * The total timeout period is approximately 100 seconds, thus
 * compensating for a gateway rebooting (~40 seconds).
 * -RPB, 9 Feb 88
```

```
 *
 * Calculations:                                                    30
 *    total time = timeout * (2^(retrans) - 1)
 *    [derived from sum of geometric series = a(r^n-1)/(r-1)]
 *       a = initial timeout
 *       r = 2
 *       n = retrans
 *
 *    This holds true while timeout * 2^retrans <= 60 seconds
 */

#define TIMEO_DEFAULT 8                                             40
#define RETRANS_DEFAULT 7

nfs_attach(at, mopt, errorout, flags)
        struct _attachtab *at;
        struct mntopts   *mopt;
        int errorout;
        struct flags *flags;
{
        char fsname[BUFSIZ];
        static char myhostname[BUFSIZ] = "";                        50

        /*
         * Handle the 'n' mode.
         */
        if (at->mode == 'n') {
                add_options(mopt, "ro");
        }

        /*
         * Try to figure out information about myself.  Use fsname   60
         * as a temporary buffer.
         */
        if (!myhostname[0]) {
                if (gethostname(myhostname, sizeof(myhostname))) {
                        if (flags->debug_flag)
                                perror("gethostname");
                }
        }

        if (myhostname[0]) {                                        70
                if (host_compare(myhostname, at->host, flags)) {
                        if (!flags->override) {
                                fprintf(stderr,
                                        "%s:  (filesystem %s) NFS self mount not allowed.\n",
                                        config.progname, at->hesiodname);
                                flags->status = ERR_ATTACHNOTALLOWED;
                                return(FAILURE);
                        }
                        fprintf(stderr, "%s:  (filesystem %s) warning:  NFS self mount\n",
                                config.progname, at->hesiodname);   80
                }
        }
```

```
        if ((at->mode != 'n') && flags->do_nfsid)
                if (nfsid(at->host, at->hostaddr[0], MOUNTPROC_KUIDMAP,
                        errorout, at->hesiodname, 1,
                        flags->owner_uid, flags) == FAILURE) {
                        if (mopt->flags & M_RDONLY) {
                                printf("%s:  Warning, mapping failed for filesystem %s,\n\tcont
                                        config.progname, at->hesiodname);
                                /* So the mount rpc wins */
                                clear_errored(at->hostaddr[0]);
                        } else if(at->mode == 'm') {
                                printf("%s:  Warning, mapping failed for filesystem %s.\n",
                                        config.progname, at->hesiodname);
                                flags->status = 0;
                                clear_errored(at->hostaddr[0]);
                        } else
                                return (FAILURE);
                }

        if (!(mopt->tsa.nfs.flags & NFSMNT_RETRANS)) {
                mopt->tsa.nfs.flags |= NFSMNT_RETRANS;
                mopt->tsa.nfs.retrans = RETRANS_DEFAULT;
        }

        if (!(mopt->tsa.nfs.flags & NFSMNT_TIMEO)) {
                mopt->tsa.nfs.flags |= NFSMNT_TIMEO;
                mopt->tsa.nfs.timeo = TIMEO_DEFAULT;
        }

        /* XXX This is kind of bogus, because if a filesystem has a number
         * of hesiod entries, and the mount point is busy, each one will
         * be tried until the last one fails, then an error printed.
         * C'est la vie.
         */

        sprintf(fsname, "%s:%s", at->host, at->hostdir);

        if (mountfs(at, fsname, mopt, errorout, flags) == FAILURE) {
                if ((at->mode != 'n') && flags->do_nfsid)
                        nfsid(at->host, at->hostaddr[0], MOUNTPROC_KUIDUMAP,
                            errorout, at->hesiodname, 1, flags->owner_uid,
                            flags);
                return (FAILURE);
        }

        return (SUCCESS);
}

/*
 * Detach an NFS filesystem.
 */
nfs_detach(at, list, flags)
   struct _attachtab *at;
   struct attach_list *list;
```

```
    struct flags *flags;
{
        if ((at->mode != 'n') && flags->do_nfsid &&
            nfsid(at->host, at->hostaddr[0], MOUNTPROC_KUIDUMAP, 1,
                  at->hesiodname,0, flags->owner_uid, flags) == FAILURE)
                printf("%s:  Warning:  couldn't unmap filesystem %s/host %s\n",
                     config.progname, at->hesiodname, at->host);

        if (at->flags & FLAG_PERMANENT) {
                if (flags->debug_flag)
                        printf("Permanent flag on, skipping umount.\n");
                return(SUCCESS);
        }

        if (nfs_unmount(at->hesiodname, at->host, at->hostaddr[0],
                        at->mntpt, at->hostdir, flags) == FAILURE)
                return (FAILURE);

        return (SUCCESS);
}

/*
 * Parsing of explicit NFS file types
 */
int nfs_explicit(name, hesvec, hesline, flags)
    char *name, **hesvec, *hesline;
    struct flags *flags;
{
    char temp[BUFSIZ], host[BUFSIZ];
    char *dir, *cp;
    char newmntpt[BUFSIZ];

    /* XXX probably unnecessary */
    flags->filsys_type = "NFS";
    strcpy(host, name);
    dir = strchr(host, ':');
    if (!dir) {
        fprintf(stderr, "%s:  Illegal explicit definition \"%s\" for type %s\n",
                config.progname, name, flags->filsys_type);
        return -1;
    }
    *dir = '\0';
    dir++;
    if (*dir != '/') {
        fprintf(stderr, "%s:  Illegal explicit definition \"%s\" for type %s\n",
                config.progname, name, flags->filsys_type);
        return -1;
    }

    if (!flags->mntpt) {
            strcpy(temp, host);
            /*
             * Zero out any domain names, since they're ugly as mount
             * points.
```

```
		 */
		if (cp = strchr(temp, '.'))
			*cp = '\0';
		if (!strcmp(dir, "/")) {
			if (config.nfs_root_hack)
				sprintf(newmntpt, "%s/%s/root",
					config.nfs_mount_dir, temp);
			else
				sprintf(newmntpt, "%s/%s", config.nfs_mount_dir,
					temp);
		} else
			sprintf(newmntpt, "%s/%s%s", config.nfs_mount_dir, temp,
				dir);
	}

	sprintf(hesline, "NFS %s %s %c %s", dir, host, flags->override_mode ?
		flags->override_mode : 'w',
		flags->mntpt ? flags->mntpt : newmntpt);
	hesvec[0] = hesline;
	hesvec[1] = 0;
	return 0;
}
#endif
```

## A.3.15 pathcan.c

```
/*
 *	$Source: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/pathcan.c,v $
 *	$Author: miki $
 *	$Header: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/pathcan.c,v 1.4 94/03
 */

#ifndef lint
static char *rcsid_pathcan_c = "$Header:  /afs/dev.mit.edu/project/release/source/src/athena/bi
#endif

#include <stdio.h>
#include <errno.h>
#include <strings.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/stat.h>
#ifdef POSIX
#include <unistd.h>
#endif
#define MAXLINKS	16

extern int	errno;
extern char	*malloc();

int	pc_symlinkcnt;
int	pc_depthcnt;
```

```
#ifdef DEBUG
static char     *strdup();
#else                                                          30
extern char     *strdup();
#endif

struct filestack {
        char    *path;
        char    *ptr;           /* pointer to the next thing to fix */
};

char *path_canon(infile)
        char    *infile;                                       40
{
#ifndef POSIX
        extern char *getwd();
#endif
        static struct filestack     stack[MAXLINKS];
        int                         stkcnt = 0;       /* Stack counter */
        struct filestack            *stkptr = &stack[0]; /*Always = &stack[stkcnt]*/
        register int        i;
        register char       *cp, *token;
        static char         outpath[MAXPATHLEN];               50
        char    *outpathend = outpath; /* Always points to the end of outpath*/
        char                bfr[MAXPATHLEN];
        struct  stat        statbuf;
        int                 errorflag = 0;

        pc_symlinkcnt = pc_depthcnt = 0;

        if (!infile)
                return(NULL);
                                                               60
        stkptr->ptr = stkptr->path = strdup(infile);

        if (*infile == '/') {
                /*
                 * We're starting at the root; normal case
                 */
                stkptr->ptr++;
                outpath[0]='\0';
        } else {
                /*                                             70
                 * We're being asked to interpret a relative pathname;
                 * assume this is happening relative to the current
                 * directory.
                 */
#ifdef POSIX
            if (getcwd(outpath, sizeof(outpath)) == NULL) {
#else
                if (getwd(outpath) == NULL) {
#endif
#ifdef TEST                                                    80
                        printf("getwd returned error, %s", outpath);
```

```
#endif
                return(NULL);
        }
        outpathend += strlen(outpathend);
    }

    while (stkcnt >= 0) {
        /*
         * If there's no more pathname elements in this level
         * of recursion, pop the stack and continue.
         */
        if (!stkptr->ptr || !*stkptr->ptr) {
#ifdef TEST
            printf("Popping....  stkcnt = %d\n", stkcnt);
#endif
            free(stkptr->path);
            stkcnt--;
            stkptr--;
            continue;
        }
#ifdef TEST
        printf("stkcnt = %d, ptr = %s, out = '%s'\n", stkcnt,
            stkptr->ptr, outpath);
#endif

        /*
         * Peel off the next token and bump the pointer
         */
        token = stkptr->ptr;
        if (cp = strchr(stkptr->ptr, '/')) {
            *cp = '\0';
            stkptr->ptr = cp+1;
        } else
            stkptr->ptr = NULL;

        /*
         * If the token is "" or ".", then just continue
         */
        if (!*token || !strcmp(token, "."))
            continue;

        /*
         * If the token is "..", then lop off the last part of
         * outpath, and continue.
         */
        if (!strcmp(token, "..")) {
            if (cp = strrchr(outpath, '/'))
                *(outpathend = cp) = '\0';
            continue;
        }

        /*
         * Tack on the new token, but don't advance outpathend
         * yet (we may need to back out).
```

```
                 */
                *outpathend = '/';
                (void) strcpy(outpathend+1, token);
                if (!errorflag && lstat(outpath, &statbuf)) {
#ifdef TEST                                                              140
                        if (errno)
                                perror(outpath);
#endif
                        /*
                         * If we get a file not found, or the file is
                         * not a directory, set a flag so that
                         * lstat() is skipped from being called, since
                         * there's no point in trying any future
                         * lstat()'s.
                         */                                              150
                        if (errno == ENOTDIR || errno == ENOENT) {
                                errorflag = errno;
                                outpathend += strlen(outpathend);
                                continue;       /* Go back and pop stack */
                        }
                        return(NULL);
                }

                /*
                 * If outpath expanded to a symlink, we're going to       160
                 * expand it.  This entails: 1) reading the value of
                 * the symlink.  2) Removing the appended token to
                 * outpath.  3) Recursively expanding the value of the
                 * symlink by pushing it onto the stack.
                 */
                if (!errorflag && (statbuf.st_mode & S_IFMT) == S_IFLNK) {
                        pc_symlinkcnt++;

                        if ((i = readlink(outpath, bfr, sizeof(bfr))) < 0) {
#ifdef TEST                                                              170
                                perror("readlink");
#endif
                                return(NULL);
                        }
                        bfr[i] = '\0';
#ifdef TEST
                        printf("stkcnt = %d, found symlink to %s\n",
                              stkcnt, bfr);
#endif
                        *outpathend = '\0'; /* Back it out */            180
                        stkcnt++;
                        stkptr++;
                        if (stkcnt >= MAXLINKS) {
                                errno = ELOOP;
#ifdef TEST
                                printf("Stack limit exceeded!  Aborting...\n");
#endif
                                return(NULL);
                        }
```

```
                        stkptr->ptr = stkptr->path = strdup(bfr);
                        if (bfr[0] == '/') {
                                /* This is a sym link to root, we can */
                                /* blast outpath and start over */
                                outpathend = outpath;
                                *outpath = '\0';
                                stkptr->ptr++; /* Bump past / */
                        }
                        if (stkcnt > pc_depthcnt)
                                pc_depthcnt = stkcnt;
                        continue;
                }
                /*
                 * This is a normal case.  Extend out outpathend,
                 * and continue to the next path element
                 */
                outpathend += strlen(outpathend);
        }
        /*
         * Special case: if outpath is empty, this means we're at the
         * filesystem root
         */
        if (!*outpath)
                return("/");
        else
                return(outpath);
}


#ifdef DEBUG
/*
 * Duplicate a string in malloc'ed memory
 */
static char *strdup(s)
        char    *s;
{
        register char   *cp;

        if (!(cp = malloc(strlen(s)+1))) {
                printf("Out of memory!!!\n");
                abort();
        }
        return(strcpy(cp,s));
}

main (argc, argv)
        int argc;
        char **argv;
{
        if (argc != 2) {
                fprintf(stderr, "Usage:  %s pathname\n", argv[0]);
                exit(1);
        }
        printf("Result:  %s\n", path_canon(argv[1]));
```

```
        printf("Number of symlinks traversed:  %d\n", pc_symlinkcnt);
        printf("Maximum depth of symlink traversal:  %d\n", pc_depthcnt);
        exit(0);
}
#endif
```

## A.3.16 rpc.c

```
/*      Created by:         Robert French
 *
 *      $Source: /afs/rel-eng.athena.mit.edu/project/release/current/source/athena/athena.bin/attach/RCS/rpc.c,
 *      $Author: vrt $
 *
 *      Copyright (c) 1988 by the Massachusetts Institute of Technology.
 */

static char *rcsid_rpc_c = "$Header:  /afs/rel-eng.athena.mit.edu/project/release/current/source/athe
                                                                                    10
#include "attach.h"
#ifdef NFS
#include <krb.h>
#if defined(_AIX) && (AIXV < 30)
#include <rpc/rpcmount.h>
#include <rpc/nfsmount.h>
#endif

#if defined(SOLARIS)
#include <limits.h>                                                                 20
#define NGROUPS NGROUPS_MAX
#endif

extern bool_t     xdr_void();
extern char       *krb_getrealm();

static struct cache_ent rpc_cache[RPC_MAXCACHE];
static int first_free = 0;
static pthread_mutex_t cache_lock = PTHREAD_MUTEX_INITIALIZER;
static pthread_mutex_t rpc_lock = PTHREAD_MUTEX_INITIALIZER;                        30
static pthread_mutex_t krb_lock = PTHREAD_MUTEX_INITIALIZER;

/*
 * XDR for sending a Kerberos ticket -- sends the whole KTEXT block,
 * but this is very old lossage, and nothing that can really be fixed
 * now.
 */

bool_t xdr_krbtkt(xdrs, authp)
   XDR *xdrs;                                                                       40
   KTEXT authp;
{
   KTEXT_ST auth;

   auth = *authp;
```

```
    auth.length = htonl(authp->length);
    return xdr_opaque(xdrs, (void *) &auth, sizeof(KTEXT_ST));
}

/*                                                                    50
 * Maintain a cache of RPC handles and error status.
 * If addr found, returns with cache locked.
 */

struct cache_ent *lookup_cache_ent(addr)
    struct in_addr addr;
{
    int i;

    pthread_mutex_lock(&cache_lock);                                  60
    for (i=0;i<first_free;i++)
        if (!bcmp(&addr, &rpc_cache[i].addr, sizeof(addr)))
            return (&rpc_cache[i]);

    pthread_mutex_unlock(&cache_lock);
    return (NULL);
}

void add_cache_ent(addr, handle, fd, sin)
    struct in_addr addr;                                              70
    CLIENT *handle;
    int fd;
    struct sockaddr_in *sin;
{
    int ind;

    pthread_mutex_lock(&cache_lock);

    /* We lose a tiny bit of efficiency if we overflow the cache */
    if (first_free == RPC_MAXCACHE) {                                 80
        ind = 0;
        pthread_mutex_lock(&rpc_lock);
        clnt_destroy(rpc_cache[0].handle);
        pthread_mutex_unlock(&rpc_lock);
        close(rpc_cache[0].fd);
    }
    else
        ind = first_free++;
    rpc_cache[ind].addr = addr;
    rpc_cache[ind].handle = handle;                                   90
    rpc_cache[ind].fd = fd;
    rpc_cache[ind].sin = *sin;
    rpc_cache[ind].error = 0;

    pthread_mutex_unlock(&cache_lock);
}

int errored_out(addr)
    struct in_addr addr;
```

```
{                                                                    100
    struct cache_ent *ent;
    int error;

    if (ent = lookup_cache_ent(addr)) {
        error = ent->error;
        pthread_mutex_unlock(&cache_lock);
        return (error);
    }
    return (0);
}                                                                    110

int mark_errored(addr)
    struct in_addr addr;
{
    struct cache_ent *ent;

    if (ent = lookup_cache_ent(addr)) {
        ent->error = 1;
        pthread_mutex_unlock(&cache_lock);
    }                                                                120
}

int clear_errored(addr)
    struct in_addr addr;
{
    struct cache_ent *ent;

    if (ent = lookup_cache_ent(addr)) {
        ent->error = 0;
        pthread_mutex_unlock(&cache_lock);                           130
    }
}

/*
 * Create an RPC handle for the given address.  Attempt to use one
 * already cached if possible.
 */

CLIENT *rpc_create(addr, sinp, flags)
    struct in_addr addr;                                             140
    struct sockaddr_in *sinp;
    struct flags *flags;
{
    CLIENT *client;
    struct sockaddr_in sin, sin2;
    struct timeval timeout;
    struct cache_ent *ent;
    int s;
    int err;
    u_short port;                                                    150

    if (flags->debug_flag)
        printf("Getting RPC handle for %s\n", inet_ntoa(addr));
```

```
if (ent = lookup_cache_ent(addr)) {
        if (flags->debug_flag)
            printf("Using cached entry!  FD = %d Error = %d\n", ent->fd,
                    ent->error);
        if (ent->error) {
            /* Error status will already be set */                          160
            client = NULL;
            return ((CLIENT *)0);
        }
        else {
            *sinp = ent->sin;
            client = ent->handle;
        }
        pthread_mutex_unlock(&cache_lock);
        return (client);
}                                                                           170

sin.sin_family = AF_INET;
sin.sin_addr = addr;
sin.sin_port = 0;
s = RPC_ANYSOCK;
timeout.tv_usec = 0;
timeout.tv_sec = 20;

pthread_mutex_lock(&rpc_lock);
if (!(client = clntudp_create(&sin, MOUNTPROG, MOUNTVERS,                   180
                                    timeout, &s))) {
        pthread_mutex_unlock(&rpc_lock);
        if (flags->debug_flag)
            printf("clntudp_create failed!\n");
        add_cache_ent(addr, 0, 0, sinp);
        mark_errored(addr);
        flags->status = ERR_HOST;
        return (NULL);
}
pthread_mutex_unlock(&rpc_lock);                                            190

*sinp = sin;

get_myaddress(&sin2);
sin2.sin_family = AF_INET;
err = 1;
for (port=IPPORT_RESERVED-1;err && port >= IPPORT_RESERVED/2;
        port--) {
        sin2.sin_port = htons(port);
        err = bind(s, (struct sockaddr *) &sin2, sizeof(sin2));             200
        if (err) {
                err = errno;
                if (err == EINVAL) {
                        /*
                         * On the NeXT, and possibly other Mach machines,
                         * the socket from clntudp_create is already bound.
                         * So if it is, we'll just go on ahead.
```

```
                        */
                        err = 0;
                        break;
                }
        }
    }

    add_cache_ent(addr, client, s, sinp);
    if (flags->debug_flag)
        printf("Adding cache entry with FD = %d\n", s);

    if (err) {
        if (flags->debug_flag) {
                printf("Couldn't allocate a reserved port!\n");
                errno = err;
                perror("bind");
        }
        pthread_mutex_lock(&rpc_lock);
        clnt_destroy(client);
        pthread_mutex_unlock(&rpc_lock);
        client = (CLIENT *)0;
        mark_errored(addr);
        flags->status = ERR_NOPORTS;
    }

    return (client);
}

/*
 * spoofunix_create_default -- varient of authunix_create_default, with
 *      a neat trick....
 *
 * Written by Mark W. Eichin <eichin@athena.mit.edu>
 */

/* In our case, NGRPS (for RPC) is 8, but NGROUPS (kernel) is 16.  So
 * getgroups fails causing abort() if the user is in >8 groups.  We
 * ask for all groups, but only pass on the first 8. */

AUTH *
spoofunix_create_default(spoofname, uid)
        char *spoofname;
        int uid;
{
        register int len;
        char machname[MAX_MACHINE_NAME + 1];
        register gid_t gid;
        gid_t gids[NGROUPS];

        if (spoofname)
                (void) strncpy(machname, spoofname, MAX_MACHINE_NAME);
        else
                if (gethostname(machname, MAX_MACHINE_NAME) == -1) {
                        perror("gethostname");
```

```
                        abort();
                }
        machname[MAX_MACHINE_NAME] = 0;
        gid = getegid();
        if ((len = getgroups(NGROUPS,gids)) < 0) {
                fprintf(stderr,"%s:  Fatal error:  User in too many groups!\n",
                        config.progname);
                exit(1);
        }                                                                    270
        return (authunix_create(machname, uid, gid,
                                (len > NGRPS) ? NGRPS : len, gids));
}

/*
 * Perform a mapping operation of some kind on the indicated host.  If
 * errorout is zero, don't print error messages.  errname is the name
 * of the filesystem/host/whatever to prefix error message with.
 */
                                                                     280
int nfsid(host, addr, op, errorout, errname, inattach, uid, flags)
   char *host;
   struct in_addr addr;
   int op, errorout;
   char *errname;
   int inattach, uid;
   struct flags *flags;
{
   CLIENT *client;
   KTEXT_ST authent;                                                 290
   int status;
   struct timeval timeout;
   struct sockaddr_in sin;
   enum clnt_stat rpc_stat;
   char instance[INST_SZ+REALM_SZ+1];
   char *realm;
   char *ptr;

   if (flags->debug_flag)
        printf("nfsid operation on %s, op = %d\n", inet_ntoa(addr), op);     300

   /*
    * Check for previous error condition on this host
    */
   if (errored_out(addr)) {
        if (flags->debug_flag)
           printf("Host previously errored - ignoring\n");
        if (errorout)
           fprintf(stderr, "%s:  Ignoring host %s for filesystem %s due to previous host errors\
                config.progname, host, errname);                          310
        /*
         * Don't bother setting error status...we already will have
         * before
         */
        return (FAILURE);
```

```
    }

#ifdef KERBEROS
    /*
     * Mapping and user purging are the only authenticated functions...
     */
    if (op == MOUNTPROC_KUIDMAP || op == MOUNTPROC_KUIDUPURGE) {
        if (flags->debug_flag)
            printf("nfsid operation requiring authentication...op = %d\n",
                op);
        pthread_mutex_lock(&krb_lock);
        realm = (char *) krb_realmofhost(host);
        pthread_mutex_unlock(&krb_lock);
        strcpy(instance, host);
        ptr = strchr(instance, '.');
        if (ptr)
            *ptr = '\0';
        for (ptr=instance;*ptr;ptr++)
            if (isupper(*ptr))
                *ptr = tolower(*ptr);

        if (flags->debug_flag)
            printf("krb_mk_req for instance %s, realm %s\n", instance, realm);

        pthread_mutex_lock(&krb_lock);
        status = krb_mk_req(&authent, KERB_NFSID_INST, instance,
                            realm, 0);
        pthread_mutex_unlock(&krb_lock);
        if (status != KSUCCESS) {
            if (flags->debug_flag)
                printf("krb_mk_req failed!  status = %d\n", status);
            if (status == KDC_PR_UNKNOWN) {
              if(inattach) {
                fprintf(stderr,
                        "%s:  (warning) Host %s isn't registered with kerberos\n",
                        config.progname, host);
                fprintf(stderr, "\tmapping failed for filesystem %s.\n",
                        errname);
                return(SUCCESS);
              } else {
                fprintf(stderr, "%s:  Host %s isn't registered with kerberos\n",
                        config.progname, host);
              }
            } else {
              if (errorout)
                {
                 fprintf(stderr, "%s:  Could not get Kerberos ticket (%s.%s@%s)\n\tfor filesystem %s,
                         config.progname, KERB_NFSID_INST, instance, realm,
                         errname, krb_err_txt[status]);
                }
            }
            flags->status = ERR_KERBEROS;
            return (FAILURE);
        }
```

```
    }
#endif

    /*
     * Get an RPC handle
     */
    if ((client = rpc_create(addr, &sin, flags)) == NULL) {
            if (errorout)
                fprintf(stderr, "%s:  server %s not responding (for filesystem %s)\n",
                        config.progname, host, errname);
            /*
             * Error status set by rpc_create
             */
            return (FAILURE);
    }

    client->cl_auth = spoofunix_create_default(flags->spoofhost, uid);

    timeout.tv_usec = 0;
    timeout.tv_sec = 20;

    pthread_mutex_lock(&rpc_lock);
    if (op == MOUNTPROC_KUIDMAP || op == MOUNTPROC_KUIDUPURGE)
            rpc_stat = clnt_call(client, op, xdr_krbtkt, (char *) &authent,
                                 xdr_void, 0, timeout);
    else
            rpc_stat = clnt_call(client, op, xdr_void, 0, xdr_void,
                                 0, timeout);
    pthread_mutex_unlock(&rpc_lock);
    if (rpc_stat != RPC_SUCCESS) {
            if (errorout) {
                switch (rpc_stat) {
                case RPC_TIMEDOUT:
                        fprintf(stderr, "%s:  timeout while contacting mount daemon on %s (for filesyste
                                config.progname, host, errname);
                        if (inattach)
                            fprintf(stderr, "\twhile mapping - try attach -n\n",
                                    errname);
                        flags->status = ERR_HOST;
                        break;
                case RPC_AUTHERROR:
                        fprintf(stderr, "%s:  Authentication failed to host %s for filesystem %s\n",
                                config.progname, host, errname);
                        flags->status = ERR_AUTHFAIL;
                        break;
                case RPC_PMAPFAILURE:
                        fprintf(stderr, "%s:  Can't find mount daemon on %s for filesystem %s\n",
                                config.progname, host, errname);
                        flags->status = ERR_HOST;
                        break;
                case RPC_PROGUNAVAIL:
                case RPC_PROGNOTREGISTERED:
                        fprintf(stderr, "%s:  Mount daemon not available on %s (filesystem %s)\n",
                                config.progname, host, errname);
```

```
                flags->status = ERR_HOST;
                break;
            case RPC_PROCUNAVAIL:
                fprintf(stderr, "%s: Warning: mount daemon on %s doesn't understand UID maps\n\t(fil
                    config.progname, host, errname);
                return(SUCCESS);
            default:                                                          430
                fprintf(stderr, "%s: System error contacting server %s for filesystem %s\n",
                        config.progname, host, errname);
                flags->status = ERR_HOST;
                break;
            }
        }
        mark_errored(addr);
        if (flags->debug_flag) {
            pthread_mutex_lock(&rpc_lock);
            clnt_perror(client, "RPC return status");                         440
            pthread_mutex_unlock(&rpc_lock);
        }
        return (FAILURE);
    }
    return (SUCCESS);
}
#endif
```

## A.3.17 strtok.c

```
#ifdef NEED_STRTOK
/* strtok.c -- return tokens from a string, NULL if no token left */
/* LINTLIBRARY */

/*
 * Get next token from string s1 (NULL on 2nd, 3rd, etc. calls),
 * where tokens are nonempty strings separated by runs of
 * chars from s2.  Writes NULs into s1 to end tokens.  s2 need not
 * remain constant from call to call.
 *                                                                          10
 * Written by reading the System V Interface Definition, not the code.
 *
 * Totally public domain.
 *
 */

#define NULL  0

char *
strtok(s1, s2)                                                               20
char *s1;
register char *s2;
{
        register char *scan;
        char *tok;
        register char *scan2;
```

```
        static char *scanpoint = (char *)NULL;

        if (s1 == (char *)NULL && scanpoint == (char *)NULL)
                return((char *)NULL);                                    30
        if (s1 != (char *)NULL)
                scan = s1;
        else
                scan = scanpoint;

        /*
         * Scan leading delimiters.
         */
        for (; *scan != '\0'; scan++) {
                for (scan2 = s2; *scan2 != '\0'; scan2++)                40
                        if (*scan == *scan2)
                                break;
                if (*scan2 == '\0')
                        break;
        }
        if (*scan == '\0') {
                scanpoint = (char *)NULL;
                return((char *)NULL);
        }
                                                                         50
        tok = scan;

        /*
         * Scan token.
         */
        for (; *scan != '\0'; scan++) {
                for (scan2 = s2; *scan2 != '\0';)        /* ++ moved down. */
                        if (*scan == *scan2++) {
                                scanpoint = scan+1;
                                *scan = '\0';                            60
                                return(tok);
                        }
        }

        /*
         * Reached end of string.
         */
        scanpoint = (char *)NULL;
        return(tok);
}                                                                        70

/* strtok.c ends here */
#endif /* NEED_STRTOK */
```

## A.3.18 ufs.c

```
/*      Created by:  Theodore Ts'o
 *
 *      $Source: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/ufs.c,v $
```

```
 *	$Author: miki $
 *
 *	Copyright (c) 1988 by the Massachusetts Institute of Technology.
 */

#ifndef lint
static char rcsid_ufs_c[] = "$Header:  /afs/dev.mit.edu/project/release/source/src/athema/bin/attach/RC
#endif

#include "attach.h"
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/file.h>

static int perform_fsck _P((char *device, char *errorname, int useraw,
                            struct flags *flags));

#ifdef UFS
/*
 * Attach an Unix filesystem
 */

int ufs_attach(at, mopt, errorout, flags)
        struct _attachtab *at;
        struct mntopts  *mopt;
        int errorout;
        struct flags *flags;
{
   if (at->mode == 'w' && !flags->skip_fsck) {
           if (perform_fsck(at->hostdir, at->hesiodname, 1, flags) == FAILURE)
                   return(FAILURE);
   }

   /* XXX This is kind of bogus, because if a filesystem has a number
    * of hesiod entries, and the mount point is busy, each one will
    * be tried until the last one fails, then an error printed.
    * C'est la vie.
    */

   if (mountfs(at, at->hostdir, mopt, errorout, flags) == FAILURE) {
        return (FAILURE);
   }

   return (SUCCESS);
}

/*
 * Detach a Unix filesystem
 */
int ufs_detach(at, list, flags)
   struct _attachtab *at;
   struct attach_list *list;
   struct flags *flags;
{
```

```
    if (at->flags & FLAG_PERMANENT) {
        if (flags->debug_flag)
            printf("Permanent flag on, skipping umount.\n");
        return(SUCCESS);
    }

    if (unmount_42(at->hesiodname, at->mntpt, at->hostdir) == FAILURE)
        return (FAILURE);

    return (SUCCESS);
}

/*
 * Parsing of explicit UFS file types
 */
int ufs_explicit(name, hesvec, hesline, flags)
    char *name, **hesvec, *hesline;
    struct flags *flags;
{
    sprintf(hesline, "UFS %s %c %s", name, flags->override_mode ?
            flags->override_mode : 'w', flags->mntpt ? flags->mntpt : "/mnt");
    hesvec[0] = hesline;
    hesvec[1] = 0;
    return(0);
}

#endif

/*
 * Attach an "error" filesystem.
 * (Print an error message and go away; always fails)
 * Not strictly UFS, but it's a good place to sneak it in...
 */
int err_attach(at, mopt, errorout, flags)
        struct _attachtab *at;
        struct mntopts   *mopt;
        int errorout;
        struct flags *flags;
{
        fprintf(stderr, "%s:  error:  %s\n", at->hesiodname, at->hostdir);
        return(FAILURE);
}

#if defined(UFS) || defined(RVD)

/*
 * Subroutine to check a filesystem with fsck
 */
static int perform_fsck(device, errorname, useraw, flags)
        char     *device, *errorname;
        int      useraw;
        struct flags *flags;
{
        static char      rdevice[512];
```

```
        static char     *fsck_av[4];
        int     error_ret, save_stderr;
#ifdef POSIX
        int     waitb;
#else
        union wait      waitb;
#endif

        strncpy(rdevice, device, sizeof(rdevice));                              120

        /* Try to generate the raw device, since it's almost always faster */
        if (useraw) {
                char    *cpp, *cpp2;
                struct stat     buf;

                if (!(cpp = strrchr(rdevice, '/')))
                        cpp = rdevice;
                else
                        cpp++;                                                  130
                *cpp++ = 'r';
                if (!(cpp2 = strrchr(device, '/')))
                        cpp2 = device;
                else
                        cpp2++;
                (void) strcpy(cpp, cpp2);

                /*
                 * Try to stat the constructed rdevice.  If it isn't a
                 * device file or if it doesn't exist, give up and use          140
                 * the original file.
                 */
                if (stat(rdevice,&buf) || !(buf.st_mode & (S_IFCHR|S_IFBLK)))
                        strcpy(rdevice,device);
        }

        if (flags->debug_flag)
                printf("performing an fsck on %s\n", rdevice);

        fsck_av[0] = FSCK_SHORTNAME;                                            150
        fsck_av[1] = "-p";                      /* Preen option */
        fsck_av[2] = rdevice;
        fsck_av[3] = NULL;
        switch(vfork()) {
        case -1:
                perror("vfork:  to fsck");
                flags->status = ERR_ATTACHFSCK;
                return(FAILURE);
        case 0:
                if (!flags->debug_flag) {                                       160
                        save_stderr = dup(2);
                        close(0);
                        close(1);
                        close(2);
                        open("/dev/null", O_RDWR);
```

```
                dup(0);
                dup(0);
        }

        execv(FSCK_FULLNAME, fsck_av);                                  170
        if (!flags->debug_flag)
                dup2(save_stderr, 2);
        perror(FSCK_FULLNAME);
        exit(1);
        /*NOTREACHED*/
default:
        if (wait(&waitb) < 0) {
                perror("wait:  for fsck");
                flags->status = ERR_ATTACHFSCK;
                return(FAILURE);                                        180
        }
}

#ifdef POSIX
        if ((error_ret = WEXITSTATUS(waitb))) {
#else
        if (error_ret = waitb.w_retcode) {
#endif
                fprintf(stderr,
                        "%s:  fsck returned a bad exit status (%d)\n",  190
                        errorname, error_ret);
                flags->status = ERR_ATTACHFSCK;
                return (FAILURE);
        }
        return(SUCCESS);
}

#endif
```

## A.3.19 unmount.c

```
/*
 * $Id: unmount.c,v 1.10 94/06/20 14:42:14 vrt Exp $
 *
 * Copyright (c) 1988,1991 by the Massachusetts Institute of Technology.
 *
 * For redistribution rights, see "mit-copyright.h"
 */

static char *rcsid_mount_c = "$Header:  /afs/dev.mit.edu/project/release/source/src/athena/bin/
                                                                        10
#include "attach.h"


#if !defined(ultrix) && !defined(_IBMR2) && !defined(SOLARIS)
#include <mntent.h>
#endif
```

```
#if defined(_AIX) && (AIXV < 30)
#include <rpc/nfsmount.h>
#include <rpc/rpcmount.h>                                                    20
#endif

#ifdef _AIX
#define unmount(x) umount(x)
#endif

#ifdef _IBMR2
#include <sys/id.h>
#endif
                                                                             30

/*
 * Unmount a filesystem.
 */
unmount_42(errname, mntpt, dev)
        char *errname;
        char *mntpt;
        char *dev;
{
#ifdef UMOUNT_CMD                                                            40
        int status;

    if (setuid(0)) {
                fprintf(stderr,"%s:  unable to change the uid to 0\n", errname);
                return(FAILURE);
        }

        switch (fork()) {
        case -1:
                fprintf(stderr, "%s:  unable to fork\n", errname);           50
                return(FAILURE);
                /* NOTREACHED */
        case 0:
                execl(UMOUNT_CMD, UMOUNT_CMD, mntpt, (char *)0);
                exit(1);
                /* NOTREACHED */
        default:
                wait(&status);
                break;
        }                                                                    60
#if !defined(SOLARIS) && !defined(linux)
        return(status ? FAILURE : SUCCESS);
#else
        return(status == 0 || ! is_mountpoint (mntpt) ? SUCCESS : FAILURE);
#endif
#else /* !UMOUNT_CMD */

#if defined(_AIX) && (AIXV > 30)
#include <sys/fullstat.h>
                                                                             70
   struct stat statb;
```

```
    if (statx(mntpt, &statb, 0, STX_NORMAL) < 0) {
        fprintf(stderr,
                "%s:  Directory %s appears to have already been removed\n",
                errname, mntpt);
        return(SUCCESS);
    }
    if ((statb.st_flag & FS_MOUNT) == 0) {
        fprintf(stderr,                                                    80
                "%s:  Directory %s is no longer a mount point\n",
                errname, mntpt);
        return(SUCCESS);
    }
    if (uvmount(statb.st_vfs, 0)) {
        if (errno == EINVAL || errno == ENOENT) {
            fprintf(stderr,
                    "%s:  Directory %s appears to already be unmounted\n",
                    errname, mntpt);
            return(SUCCESS);                                               90
        } else {
            fprintf(stderr, "%s:  Unable to unmount %s:  %s\n", errname,
                    mntpt, sys_errlist[errno]);
            return (FAILURE);
        }
    }
    return(SUCCESS);

#else /* !AIX 3.1 */
                                                                          100
        FILE *tmpmnt, *mnted;
        char *tmpname;
#ifndef ultrix
        struct mntent *mnt;
#endif
        int tmpfd;

#ifdef ultrix
        int fsloc = 0, found = 0;
        struct fs_data fsdata;                                            110

        while (getmountent(&fsloc, &fsdata, 1) > 0) {
                if (!strcmp(fsdata.fd_path, mntpt)) {
                        found = 1;
                        break;
                }
        }
        if (!found) {
                fprintf(stderr,
                        "%s:  Directory %s appears to already be unmounted\n",    120
                        errname, mntpt);
                return(SUCCESS);
        }
        /* this hack to avoid ugly ifdef's */
#define unmount(x) umount(fsdata.fd_dev)
```

```
#endif /* ultrix */

#if defined(_AIX) && (AIXV < 30)
        if (unmount(dev ? dev : mntpt) < 0)
#else                                                                   130
        if (unmount(mntpt) < 0)
#endif
        {
                if (errno == EINVAL || errno == ENOENT
#ifdef _AIX
                   || errno == ENOTBLK
#endif
                   ) {
                        fprintf(stderr,
                                "%s:  Directory %s appears to already be unmounted\n",
                                errname, mntpt);
                        /* Continue on, to flush mtab if necessary */
                } else {
                        fprintf(stderr, "%s:  Unable to unmount %s:  %s\n", errname,
                                mntpt, sys_errlist[errno]);
                        return (FAILURE);
                }
        }

#ifdef ultrix                                                           150
#undef unmount
        return(SUCCESS);
#else                                           /* !ultrix */

        lock_mtab();
        if (!(tmpname = malloc(strlen(mtab_fn)+7))) {
                fprintf(stderr, "Can't malloc temp filename for unmount!\n");
                exit(ERR_FATAL);
        }
        (void) strcpy(tmpname, mtab_fn);                                160
        (void) strcat(tmpname, "XXXXXX");
        mktemp(tmpname);
        if ((tmpfd = open(tmpname, O_RDWR|O_CREAT|O_TRUNC, 0644)) < 0) {
                fprintf(stderr, "Can't open temporary file for umount!\n");
                exit(ERR_FATAL);
        }
        close(tmpfd);
        tmpmnt = setmntent(tmpname, "w");
        if (!tmpmnt) {
                fprintf(stderr,                                         170
                        "Can't open temporary file for writing in umount!\n");
                exit(ERR_FATAL);
        }
        mnted = setmntent(mtab_fn, "r");
        if (!mnted) {
                fprintf(stderr, "Can't open %s for read:%s\n", mtab_fn,
                        sys_errlist[errno]);
                exit(ERR_FATAL);
        }
```

```
        /* Delete filesystem from /etc/mtab */                                  180
        while (mnt = getmntent(mnted))
                if (strcmp(mnt->mnt_dir, mntpt))
                        addmntent(tmpmnt, mnt);

        endmntent(tmpmnt);
        endmntent(mnted);
        if (rename(tmpname, mtab_fn) < 0) {
                fprintf(stderr, "Unable to rename %s to %s:  %s\n", tmpname,
                        mtab_fn, sys_errlist[errno]);
                exit(ERR_FATAL);                                                190
        }
        unlock_mtab();

        return (SUCCESS);
#endif /* ultrix */
#endif /* !AIX 3.1 */
#endif /* !UMOUNT_CMD */
}

#ifdef NFS                                                                      200
/*
 * Unmount an NFS filesystem
 */
nfs_unmount(errname, host, hostaddr, mntpt, rmntpt, flags)
    char *errname;
    char *host;
    struct in_addr hostaddr;
    char *mntpt;
    char *rmntpt;
    struct flags *flags;                                                        210
{
    static struct sockaddr_in sin;
    struct timeval timeout;
    CLIENT *client;
    enum clnt_stat rpc_stat;

    if (unmount_42(errname, mntpt, NULL) == FAILURE)
        return (FAILURE);

    /*                                                                          220
     * If we can't contact the host, don't bother complaining;
     * it won't actually hurt anything except that hosts rmtab.
     */
    if (errored_out(hostaddr))
        return (SUCCESS);

    if ((client = (CLIENT *)rpc_create(hostaddr, &sin, flags)) == NULL) {
        fprintf(stderr,
                "%s:  Server %s not responding\n",
                errname, host);                                                 230
        return (SUCCESS);
    }
```

```
    client->cl_auth = spoofunix_create_default(flags->spoofhost,
                                                config.real_uid);

    timeout.tv_usec = 0;
    timeout.tv_sec = 20;
    rpc_stat = clnt_call(client, MOUNTPROC_UMNT, xdr_path, (char *) &rmntpt,
                         xdr_void, NULL, timeout);                          240
    if (rpc_stat != RPC_SUCCESS) {
        mark_errored(hostaddr);
        switch (rpc_stat) {
        case RPC_TIMEDOUT:
            fprintf(stderr, "%s:  Timeout while contacting mount daemon on %s\n",
                    errname, host);
            break;
        case RPC_AUTHERROR:
            fprintf(stderr, "%s:  Authentication failed\n",
                    errname, host);                                         250
            break;
        case RPC_PMAPFAILURE:
            fprintf(stderr, "%s:  Can't find mount daemon on %s\n",
                    errname, host);
            break;
        case RPC_PROGUNAVAIL:
        case RPC_PROGNOTREGISTERED:
            fprintf(stderr, "%s:  Mount daemon not available on %s\n",
                    errname, host);
            break;                                                          260
        default:
            fprintf(stderr, "%s:  System error contacting server %s\n",
                    errname, host);
            break;
        }
        if (flags->debug_flag)
            clnt_perror(client, "RPC return status");
        return (SUCCESS);
    }
                                                                            270
    return (SUCCESS);
}
#endif

#ifdef SOLARIS
bool_t
xdr_path(xdrs, pathp)
        XDR *xdrs;
        char **pathp;
{                                                                           280
        if (xdr_string(xdrs, pathp, 1024)) {
                return(TRUE);
        }
        return(FALSE);
}

xdr_fhstatus(xdrs, fhsp)
```

```
        XDR *xdrs;
        struct fhstatus *fhsp;
{                                                                    290
        if (!xdr_int(xdrs, &fhsp->fhs_status))
                return FALSE;
        if (fhsp->fhs_status == 0) {
                if (!xdr_fhandle(xdrs, &fhsp->fhs_fh))
                        return FALSE;
        }
}
xdr_fhandle(xdrs, fhp)
        XDR *xdrs;
        fhandle_t *fhp;                                              300
{
        if (xdr_opaque(xdrs, (char *) fhp, NFS_FHSIZE)) {
                return (TRUE);
        }
        return (FALSE);
}

#endif

#if defined(SOLARIS) || defined(linux)                               310
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>

int is_mountpoint (dirname)
   char *dirname;
{
   struct stat rootstat, pointstat;
   char *parent;
   int len;                                                          320

   parent = strdup (dirname);
   len = strlen (parent) - 1;
   while (dirname[len] == '/') len--;
   while (len > 0 && dirname[len] != '/') len--;
   len++;
   parent[len] = 0;

   if (stat (parent, &rootstat) < 0) {
    return (FALSE);                                                  330
   }
   if (stat (dirname, &pointstat) < 0) {
    return (FALSE);
   }
   free (parent);
   return (rootstat.st_dev != pointstat.st_dev);
}
#endif
```

## A.3.20 util.c

```
/*      Created by:     Robert French
 *
 *      $Source: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/util.c,v $
 *      $Author: miki $
 *
 *      Copyright (c) 1988 by the Massachusetts Institute of Technology.
 */

static char *rcsid_util_c = "$Header:  /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/R
                                                                                              10
#include "attach.h"

#include <sys/stat.h>
#include <fcntl.h>
#include <pwd.h>
#include <grp.h>
#include <signal.h>
#ifdef HESIOD
#include <hesiod.h>
#endif                                                                                        20

#define TOKSEP " \t\r\n"

#ifdef ultrix
#define max(a,b) (((a) > (b)) ? (a) : (b))
#endif

char exp_hesline[BUFSIZ];           /* Place to store explicit */
char *exp_hesptr[2];                /* "hesiod" entry */
char *abort_msg = "Operation aborted\n";                                                      30

static void missarg _P((char *arg));

/*
 * These routines provide a way of locking out interrupts during
 * critical sections of code.  After the critical sections of code are
 * executed, if a SIGTERM or SIGINT had arrived before, the program
 * terminates then.
 */
                                                                                              40
int                     caught_signal = 0;
static pthread_mutex_t  critical_lock = PTHREAD_MUTEX_INITIALIZER;
static int              critical_code_segments = 0;

#ifdef POSIX
static sigset_t     osigmask;
#else
static int          osigmask;
#endif
                                                                                              50
/*
 * Signal handler for SIGTERM & SIGINT
```

```
 */
sig_catch sig_trap()
{
    if (critical_code_segments > 0) {
        caught_signal = 1;
        return;
    }
    terminate_program();                                          60
}

/*
 * Enter critical section of code
 */
void start_critical_code()
{
    pthread_mutex_lock(&critical_lock);
    critical_code_segments++;
    pthread_mutex_unlock(&critical_lock);                         70
}

/*
 * Exit critical section of code
 */
void end_critical_code()
{
    pthread_mutex_lock(&critical_lock);
    if (--critical_code_segments == 0) {
        if (caught_signal)                                        80
            terminate_program();
    }
    pthread_mutex_unlock(&critical_lock);
}

/*
 * terminate the program
 */
void terminate_program()
{                                                                 90
    exit(ERR_INTERRUPT);
}

/*
 * LOCK the mtab - wait for it if it's already locked
 */

/*
 * Unfortunately, mount and umount don't honor lock files, so these
 * locks are only valid for other attach processes.              100
 */

static int mtab_lock_fd = -1;
static pthread_mutex_t mtab_lock = PTHREAD_MUTEX_INITIALIZER;

void lock_mtab()
```

```
{
        char    *lockfn;
#ifdef POSIX
        struct flock fl;                                                    110
#endif

        pthread_mutex_lock(&mtab_lock);
        if (mtab_lock_fd < 0) {
                if (!(lockfn = malloc(strlen(config.mtab_fn)+6))) {
                        fprintf(stderr, "Can't malloc lockfile filename.\n");
                        fprintf(stderr, abort_msg);
                        exit(ERR_FATAL);
                }
                (void) strcpy(lockfn, config.mtab_fn);                      120
                (void) strcat(lockfn, ".lock");
                mtab_lock_fd = open(lockfn, O_CREAT|O_RDWR, 0644);
                if (mtab_lock_fd < 0) {
                        fprintf(stderr,"Can't open %s:  %s\n", lockfn,
                                strerror(errno));
                        fprintf(stderr, abort_msg);
                        exit(ERR_FATAL);
                }
        }
#ifdef POSIX                                                                130
        fl.l_type = F_WRLCK;
        fl.l_whence = SEEK_SET;
        fl.l_start = 0;
        fl.l_len = 0;
        fl.l_pid = getpid();
        fcntl(mtab_lock_fd, F_SETLKW, &fl);
#else
        flock(mtab_lock_fd, LOCK_EX);
#endif
}                                                                           140

/*
 * UNLOCK the mtab
 */
void unlock_mtab()
{
        close(mtab_lock_fd);
        mtab_lock_fd = -1;
        pthread_mutex_unlock(&mtab_lock);
}                                                                           150

/*
 * Convert a string type to a filesystem type entry
 */
struct _fstypes *get_fs(s)
   char *s;
{
   int i;

   if (s && *s) {                                                           160
```

```
            for (i=0;fstypes[i].name;i++) {
                    if (!strcasecmp(fstypes[i].name, s))
                            return (&fstypes[i]);
            }
    }
    return (NULL);
}

/*
 * Build a Hesiod line either from a Hesiod query or internal frobbing
 * if explicit is set.
 */
int build_hesiod_line(name, hesptr, heslen, hesline, flags)
    char *name;
    char **hesptr;
    int heslen;
    char *hesline;
    struct flags *flags;
{
    char                **realhes;
    struct _fstypes     *fsp;
    int                 result;

    if (!flags->explicit) {
            result = conf_filsys_resolve(name, hesptr);
#ifdef HESIOD
            if (result != 0 || !*hesptr)
                    result = hes_resolve_r(name, "filsys", hesptr, heslen);
#endif
            if (result != 0 || !*hesptr)
                    fprintf(stderr, "%s:  Can't resolve name\n", name);
            return (result);
    }

    fsp = get_fs(flags->filsys_type ? flags->filsys_type : "NFS");
    if (!fsp)
            return -1;
    if (!fsp->explicit) {
            fprintf(stderr, "Explicit definitions for type %s not allowed\n",
                    fsp->name);
            return -1;
    }

    return ((fsp->explicit)(name, hesptr, hesline, flags));
}

/*
 * Parse a Hesiod record
 *
 * Returns 0 on success, -1 on failure
 */
int parse_hes(hes, at, errorname)
    char *hes, *errorname;
    struct _attachtab *at;
```

```
{
    char            *cp, *t;
    struct hostent *hent;
    struct hostent_answer answer;
    int             type;
                                                                    220
    memset(at, 0, sizeof(struct _attachtab));

    if (!*hes)
            goto bad_hes_line;


    at->fs = get_fs(strtok(hes, TOKSEP));
    if (!at->fs)
            goto bad_hes_line;
    type = at->fs->type;                                            230

    if (type & TYPE_ERR) {
            strncpy(at->hostdir, strtok(NULL, ""), sizeof(at->hostdir));
            return(0);
    }

    if (!(cp = strtok(NULL, (type & TYPE_MUL) ? "" : TOKSEP)))
            goto bad_hes_line;
    strcpy(at->hostdir, cp);
                                                                    240
    if (type & ~(TYPE_UFS | TYPE_AFS | TYPE_MUL)) {
            if (!(cp = strtok(NULL, TOKSEP)))
                    goto bad_hes_line;
            strcpy(at->host, cp);
    } else
            strcpy(at->host, "localhost");

    if (type & TYPE_MUL) {
            t = at->hostdir;
            while (t = strchr(t,' '))                               250
                    *t = ',';
            at->mode = '-';
            strcpy(at->mntpt, "localhost");
    } else {

            if (!(cp = strtok(NULL, TOKSEP)))
                    goto bad_hes_line;
            at->mode = *cp;
            if (at->mode == 'x')
                    at->mode = 'w'; /* Backwards compatibility */   260
            if (at->fs->good_flags) {
                    if (!strchr(at->fs->good_flags, at->mode))
                            goto bad_hes_line;      /* Bad attach mode */
            }

            if (!(cp = strtok(NULL, TOKSEP)))
                    goto bad_hes_line;
            strcpy(at->mntpt, cp);
```

```
    }
                                                                        270
    if (type & ~(TYPE_UFS | TYPE_AFS | TYPE_MUL)) {
            hent = gethostbyname_r(at->host, &answer);
            if (!hent) {
                    fprintf(stderr,"%s:  Can't resolve host %s\n",errorname,
                            at->host);
                    fprintf(stderr, abort_msg);
                    return(-1);
            }
#ifdef POSIX
        memmove(&at->hostaddr[0].s_addr, hent->h_addr_list[0], 4);      280
#else
            bcopy(hent->h_addr_list[0], &at->hostaddr[0].s_addr, 4);
#endif
            strcpy(at->host, hent->h_name);
    } else
            at->hostaddr[0].s_addr = (long) 0;
    return(0);

bad_hes_line:
    fprintf(stderr,"Badly formatted filesystem definition\n");          290
    fprintf(stderr, abort_msg);
    return(-1);
}

/*
 * Make the directories necessary for a mount point - set the number
 * of directories created in the attachtab structure
 */
int make_mntpt(at, flags)
    struct _attachtab *at;                                              300
    struct flags *flags;
{
    char bfr[BUFSIZ],*ptr;
    int oldmask;
    struct stat statbuf;

    strcpy(bfr, at->mntpt);
    if (at->fs->flags & AT_FS_PARENTMNTPT) {
            ptr = strrchr(bfr, '/');
            if (ptr)                                                    310
                    *ptr = 0;
            else
                    return(SUCCESS);
    }

    if (!stat(bfr, &statbuf)) {
        if ((statbuf.st_mode & S_IFMT) == S_IFDIR)
            return (SUCCESS);
        fprintf(stderr, "%s:  %s is not a directory\n", at->hesiodname,
                at->mntpt);                                             320
        return (FAILURE);
    }
```

```
    oldmask = umask(022);

    ptr = bfr+1;                    /* Pass initial / */

    at->rmdir = 0;

    while (ptr && *ptr) {                                              330
        strcpy(bfr, at->mntpt);
        ptr = strchr(ptr, '/');
        if (ptr)
            *ptr++ = '\0';
        if (flags->debug_flag)
                printf("Making directory %s (%s)\n", bfr, ptr ? ptr : "");
        if (mkdir(bfr, 0777)) {
            if (errno == EEXIST)
                continue;
            fprintf(stderr, "%s:  Can't create directory %s:  %s\n",   340
                at->hesiodname, bfr, strerror(errno));
            umask(oldmask);
            return (FAILURE);
        }
        else
            at->rmdir++;
    }
    umask(oldmask);
    return (SUCCESS);
}                                                                      350

/*
 * Delete a previously main mountpoint
 */
int rm_mntpt(at, flags)
    struct _attachtab *at;
    struct flags *flags;
{
    char bfr[BUFSIZ], *ptr;
                                                                       360
    strcpy(bfr, at->mntpt);
    ptr = bfr;

    if (at->fs->flags & AT_FS_PARENTMNTPT) {
            ptr = strrchr(bfr, '/');
            if (ptr)
                    *ptr = 0;
            else
                    return(SUCCESS);
    }                                                                  370
    while (at->rmdir--) {
        if (flags->debug_flag)
                printf("Deleting directory %s (%s)\n", bfr, ptr ? ptr : "");
        if (rmdir(bfr)) {
            if (errno != ENOENT) {
                fprintf(stderr,
```

```
                        "%s:  Can't remove directory %s:  %s\n",
                        at->hesiodname, ptr,
                        strerror(errno));
                return (FAILURE);
            }
        }
        ptr = strrchr(bfr, '/');
        if (ptr)
            *ptr = '\0';
        else
            return (SUCCESS);
    }
    return (SUCCESS);
}

/*
 * Internal spiffed up getopt
 */
char internal_getopt(arg, cl)
    char *arg;
    struct command_list *cl;
{
    int i;

    for (i=0;cl[i].small;i++) {
        if (cl[i].large && !strcmp(cl[i].large, arg))
            return (cl[i].small[1]);
        if (!strcmp(cl[i].small, arg))
            return (cl[i].small[1]);
    }
    return ('?');
}

/*
 * Format a hesiod name into a name in /tmp...including replacing /
 * with @, etc.
 */
make_temp_name(filename, name)
    char *filename;
    char *name;
{
    strcpy(filename, "/tmp/attach_");
    filename = filename+strlen(filename);

    while (*name) {
        if (*name == '/')
            *filename++ = '@';
        else
            *filename++ = *name;
        name++;
    }
    *filename = '\0';
}
```

```
/*
 * Check to see if a filesystem is really being frobbed with by
 * another attach process.
 */

int really_in_use(name, flags)
    char *name;
    struct flags *flags;
{
    int fd, ret;                                                    440
    char filename[BUFSIZ];
#ifdef POSIX
    struct flock fl;
#endif

    make_temp_name(filename, name);

    if (flags->debug_flag)
        printf("Checking lock on %s\n", filename);
                                                                    450
    fd = open(filename, O_RDWR, 0644);
    if (!fd)
        return (0);

#ifdef POSIX
    fl.l_type = F_WRLCK;
    fl.l_whence = SEEK_SET;
    fl.l_start = 0;
    fl.l_len = 0;
    fl.l_pid = getpid();                                            460
    ret = (fcntl(fd, F_SETLK, &fl)==-1 && errno == EAGAIN);
#else
    ret = (flock(fd, LOCK_EX | LOCK_NB) == -1 && errno == EWOULDBLOCK);
#endif

    close(fd);
    return (ret);
}

/*                                                                  470
 * Mark a filesystem as being frobbed with
 */

void mark_in_use(name, fslock, flags)
    char *name;
    struct fslock *fslock;
    struct flags *flags;
{
#ifdef POSIX
    struct flock fl;                                                480
#endif

    if (!name) {
        if (flags->debug_flag)
```

```
            printf("Removing lock on %s\n", fslock->filename);
        close(fslock->fd);
        unlink(fslock->filename);
        return;
    }

    make_temp_name(fslock->filename, name);

    if (flags->debug_flag)
        printf("Setting lock on %s:  ", fslock->filename);

    /*
     * Unlink the old file in case someone else already has a lock on
     * it...we'll override them with our new file.
     */
    unlink(fslock->filename);
    fslock->fd = open(fslock->filename, O_CREAT|O_RDWR, 0644);
    if (!fslock->fd) {
            fprintf(stderr,"Can't open %s:  %s\n", fslock->filename,
                    strerror(errno));
            fprintf(stderr, abort_msg);
            exit(ERR_FATAL);
    }

    if (flags->debug_flag)
            printf("%d\n", fslock->fd);

#ifdef POSIX
    fl.l_type = F_WRLCK;
    fl.l_whence = SEEK_SET;
    fl.l_start = 0;
    fl.l_len = 0;
    fl.l_pid = getpid();
    fcntl(fslock->fd, F_SETLKW, &fl);
#else
    flock(fslock->fd, LOCK_EX);
#endif
}

#ifdef DEBUG
/*
 * Dump used file descriptors...useful for debugging
 */

fd_dump()
{
    int i;
    char b;

    printf("FD's in use:  ");
    for (i=0;i<64;i++)
        if (read(i,&b,0) >= 0)
            printf("%d ",i);
    printf("\n");
```

```
}
#endif

/*
 * String comparison for filesystem names - case insensitive for hesiod,
 * sensitive for explicit.
 */

int hescmp(at, s)
    struct _attachtab *at;
    char *s;
{
    if (at->explicit)
        return (strcmp(at->hesiodname, s));
    return (strcasecmp(at->hesiodname, s));
}

/*
 * Duplicate a string in malloc'ed memory
 */
char *strdup(s)
        const char      *s;
{
        register char   *cp;

        if (!(cp = malloc(strlen(s)+1))) {
                fprintf(stderr, "attach:  out of memory, exiting...\n");
                exit(1);
        }
        return(strcpy(cp,s));
}

/*
 * Check to see if we have root priv's; give an error if we don't.
 */
void check_root_privs(flags)
        struct flags *flags;
{
        if (!config.real_uid || !config.effective_uid || (flags->debug_flag))
                return;
        fprintf(stderr,
                "%s must be setuid to root for this operation to succeed.\n",
                config.progname);
        exit(ERR_FATAL);
        /* NOTREACHED */
}

void add_options(mopt, string)
        struct mntopts  *mopt;
        char            *string;
{
        char    *next, *arg, *str, *orig_str;

        orig_str = str = strdup(string);
```

```
        while (str && *str) {
                next = strchr(str, ',');
                if (next) {
                        *next++ = '\0';
                }
                arg = strchr(str, '=');
                if (arg)
                        *arg++ = '\0';                                          600
                if (!strcmp(str, "ro")) {
                        mopt->flags |= M_RDONLY;
#if defined(ultrix) && defined(NFS)
                        if (mopt->type == MOUNT_NFS) {
                           mopt->tsa.nfs.gfs_flags |= M_RONLY;
                           mopt->tsa.nfs.flags |= NFSMNT_RONLY;
                        }
#endif /* ultrix && NFS */
                } else if (!strcmp(str, "rw")) {
#ifdef ultrix                                                                   610
                   mopt->flags &= ~M_RDONLY;
                   mopt->tsa.nfs.gfs_flags &= ~M_RONLY;
                   mopt->tsa.nfs.flags &= ~NFSMNT_RONLY;
#else /* !ultrix */
                   mopt->flags &= ~M_RDONLY;
#endif /* ultrix */
                }
#ifdef ultrix
                /* process other ultrix options */
                else if (!strcmp(str, "force")) {                               620
#ifdef NFS
                   if (mopt->type == MOUNT_NFS)
                        mopt->tsa.nfs.gfs_flags |= M_FORCE;
                   else
#endif /* NFS */
                   if (mopt->type == MOUNT_UFS)
                        mopt->tsa.ufs.ufs_flags |= M_FORCE;
                } else if (!strcmp(str, "sync")) {
#ifdef NFS
                   if (mopt->type == MOUNT_NFS)                                 630
                        mopt->tsa.nfs.gfs_flags |= M_SYNC;
                   else
#endif /* NFS */
                   if (mopt->type == MOUNT_UFS)
                        mopt->tsa.ufs.ufs_flags |= M_SYNC;
                } else if (!strcmp(str, "pgthresh")) {
                   int pgthresh;

                   if (!arg) {
                        missarg("pgthresh");                                    640
                        continue;
                   }
                   pgthresh = atoi(arg);
                   pgthresh = max (pgthresh, MINPGTHRESH/PGUNITS);
#ifdef NFS
                   if (mopt->type == MOUNT_NFS) {
```

```
                        mopt->tsa.nfs.pg_thresh = pgthresh;
                        mopt->tsa.nfs.flags |= NFSMNT_PGTHRESH;
                    } else
#endif /* NFS */                                                    650
                    if (mopt->type == MOUNT_UFS)
                        mopt->tsa.ufs.ufs_pgthresh = pgthresh;
                } else if (!strcmp(str, "quota")) {
                    if (mopt->type == MOUNT_UFS)
                        mopt->tsa.ufs.ufs_flags |= M_QUOTA;
                } else if (!strcmp(str, "noexec")) {
#ifdef NFS
                    if (mopt->type == MOUNT_NFS)
                        mopt->tsa.nfs.gfs_flags |= M_NOEXEC;
                    else                                            660
#endif /* NFS */
                    if (mopt->type == MOUNT_UFS)
                        mopt->tsa.ufs.ufs_flags |= M_NOEXEC;
                } else if (!strcmp(str, "nocache")) {
#ifdef NFS
                    if (mopt->type == MOUNT_NFS)
                        mopt->tsa.nfs.gfs_flags |= M_NOCACHE;
                    else
#endif /* NFS */
                    if (mopt->type == MOUNT_UFS)                    670
                        mopt->tsa.ufs.ufs_flags |= M_NOCACHE;
                } else if (!strcmp(str, "nodev")) {
#ifdef NFS
                    if (mopt->type == MOUNT_NFS)
                        mopt->tsa.nfs.gfs_flags |= M_NODEV;
                    else
#endif /* NFS */
                    if (mopt->type == MOUNT_UFS)
                        mopt->tsa.ufs.ufs_flags |= M_NODEV;
                }                                                   680
#endif /* ultrix */
#ifndef AIX
                else if (!strcmp(str, "nosuid")) {
#ifdef ultrix
#ifdef NFS
                    if (mopt->type == MOUNT_NFS)
                        mopt->tsa.nfs.gfs_flags |= M_NOSUID;
                    else
#endif /* NFS */
                    if (mopt->type == MOUNT_UFS)                    690
                        mopt->tsa.ufs.ufs_flags |= M_NOSUID;
#else /* !ultrix */
                    mopt->flags |= M_NOSUID;
#endif /* ultrix */
                }
#endif /* AIX */
#ifdef NFS
                else if (mopt->type == MOUNT_NFS) {
                    if (!strcmp(str, "soft"))
                        mopt->tsa.nfs.flags |= NFSMNT_SOFT;         700
```

```
                        else if (!strcmp(str, "hard"))
                                mopt->tsa.nfs.flags &= ~NFSMNT_SOFT;
                        else if (!strcmp(str, "rsize")) {
                                if (!arg) {
                                        missarg("rsize");
                                }
                                mopt->tsa.nfs.rsize = atoi(arg);
                                mopt->tsa.nfs.flags |= NFSMNT_RSIZE;
                        } else if (!strcmp(str, "wsize")) {
                                if (!arg) {                                    710
                                        missarg("wsize");
                                        continue;
                                }
                                mopt->tsa.nfs.wsize = atoi(arg);
                                mopt->tsa.nfs.flags |= NFSMNT_WSIZE;
                        } else if (!strcmp(str, "timeo")) {
                                if (!arg) {
                                        missarg("timeo");
                                        continue;
                                }                                              720
                                mopt->tsa.nfs.timeo = atoi(arg);
                                mopt->tsa.nfs.flags |= NFSMNT_TIMEO;
                        } else if (!strcmp(str, "retrans")) {
                                if (!arg) {
                                        missarg("retrans");
                                        continue;
                                }
                                mopt->tsa.nfs.retrans = atoi(arg);
                                mopt->tsa.nfs.flags |= NFSMNT_RETRANS;
                        } else if (!strcmp(str, "port")) {                     730
                                if (!arg) {
                                        missarg("port");
                                        continue;
                                }
                                mopt->nfs_port = atoi(arg);
                        }
#ifdef ultrix
                        else if (!strcmp(str, "intr")) {
                           mopt->tsa.nfs.flags |= NFSMNT_INT;
                        }                                                      740
#endif /* ultrix */
                }
#endif
                str = next;
        }
        free(orig_str);
}

static void missarg(arg)
        char    *arg;                                                          750
{
        fprintf(stderr,
                "%s:  missing argument to mount option; ignoring.\n",
                arg);
```

```
}

/*
 * Return a string describing the options in the mount options structure
 */
char *stropt(mopt)                                                    760
        struct mntopts   mopt;
{
        static char      buff[BUFSIZ];
        char     tmp[80];

        buff[0] = '\0';

        if (mopt.flags & M_RDONLY)
                (void) strcat(buff, ",ro");
        else                                                          770
                (void) strcat(buff, ",rw");
#ifdef M_NOSUID
#ifdef ultrix
#ifdef NFS
        if (mopt.type == MOUNT_NFS && (mopt.tsa.nfs.gfs_flags & M_NOSUID))
           (void) strcat(buff, ",nosuid");
        else
#endif /* NFS */
        if (mopt.type == MOUNT_UFS && (mopt.tsa.ufs.ufs_flags & M_NOSUID))
           (void) strcat(buff, ",nosuid");                            780
#else /* !ultrix */
        if (mopt.flags & M_NOSUID)
                (void) strcat(buff, ",nosuid");
#endif /* ultrix */
#endif /* M_NOSUID */
#ifdef NFS
        if (mopt.type == MOUNT_NFS) {
#ifdef ultrix
                if (mopt.tsa.nfs.gfs_flags & M_FORCE)
                        (void) strcat(buff, ",force");                790
                if (mopt.tsa.nfs.gfs_flags & M_SYNC)
                        (void) strcat(buff, ",force");
                if (mopt.tsa.nfs.flags & NFSMNT_PGTHRESH) {
                        (void) strcat(buff, ",pgthresh=");
                        (void) sprintf(tmp, "%d", mopt.tsa.nfs.pg_thresh);
                        (void) strcat(buff, tmp);
                }
                if (mopt.tsa.nfs.gfs_flags & M_NOEXEC)
                        (void) strcat(buff, ",noexec");
                if (mopt.tsa.nfs.gfs_flags & M_NOCACHE)               800
                        (void) strcat(buff, ",nocache");
                if (mopt.tsa.nfs.gfs_flags & M_NODEV)
                        (void) strcat(buff, ",nodev");
                if (mopt.tsa.nfs.flags & NFSMNT_INT)
                        (void) strcat(buff, ",intr");
#endif /* ultrix */
                if (mopt.tsa.nfs.flags & NFSMNT_SOFT)
                        (void) strcat(buff, ",soft");
```

```
                if (mopt.tsa.nfs.flags & NFSMNT_RSIZE) {
                        (void) strcat(buff, ",rsize=");                          810
                        (void) sprintf(tmp, "%d", mopt.tsa.nfs.rsize);
                        (void) strcat(buff, tmp);
                }
                if (mopt.tsa.nfs.flags & NFSMNT_WSIZE) {
                        (void) strcat(buff, ",wsize=");
                        (void) sprintf(tmp, "%d", mopt.tsa.nfs.wsize);
                        (void) strcat(buff, tmp);
                }
                if (mopt.tsa.nfs.flags & NFSMNT_TIMEO) {
                        (void) strcat(buff, ",timeo=");                          820
                        (void) sprintf(tmp, "%d", mopt.tsa.nfs.timeo);
                        (void) strcat(buff, tmp);
                }
                if (mopt.tsa.nfs.flags & NFSMNT_RETRANS) {
                        (void) strcat(buff, ",retrans=");
                        (void) sprintf(tmp, "%d", mopt.tsa.nfs.retrans);
                        (void) strcat(buff, tmp);
                }
                if (mopt.nfs_port) {
                        (void) strcat(buff, ",port=");                           830
                        (void) sprintf(tmp, "%d", mopt.nfs_port);
                        (void) strcat(buff, tmp);
                }
        }
#endif /* NFS */
#ifdef ultrix
        else if (mopt.type == MOUNT_UFS) {
                if (mopt.tsa.ufs.ufs_flags & M_QUOTA)
                        (void) strcat(buff, ",quota");
                if (mopt.tsa.ufs.ufs_flags & M_FORCE)                            840
                        (void) strcat(buff, ",force");
                if (mopt.tsa.ufs.ufs_flags & M_SYNC)
                        (void) strcat(buff, ",force");
                if (mopt.tsa.ufs.ufs_pgthresh != DEFPGTHRESH) {
                        (void) strcat(buff, ",pgthresh=");
                        (void) sprintf(tmp, "%d", mopt.tsa.ufs.ufs_pgthresh);
                        (void) strcat(buff, tmp);
                }
                if (mopt.tsa.ufs.ufs_flags & M_NOEXEC)
                        (void) strcat(buff, ",noexec");                          850
                if (mopt.tsa.ufs.ufs_flags & M_NOCACHE)
                        (void) strcat(buff, ",nocache");
                if (mopt.tsa.ufs.ufs_flags & M_NODEV)
                        (void) strcat(buff, ",nodev");
            }
#endif /* ultrix */

        return(buff+1);
}                                                                                860

/*
 * Display the userid, given a uid (if possible)
```

```
 */
char *struid(uid)
        int     uid;
{
        struct  passwd  *pw;
        static char     buff[64];

        pw = getpwuid(uid);                                             870
        if (pw)
                strncpy(buff, pw->pw_name, sizeof(buff));
        else
                sprintf(buff, "#%d", uid);
        return(buff);
}

/*
 * Compare if two hosts are the same
 */                                                                     880
int host_compare(host1, host2, flags)
        char    *host1;
        char    *host2;
        struct flags *flags;
{
        char    bfr[BUFSIZ];
        static  char    last_host[BUFSIZ] = "********";
        static  struct in_addr  sin1, sin2;
        struct hostent  *host;
        struct hostent_answer answer;                                   890

        /*
         * Cache the last host1, for efficiency's sake.
         */
        if (strcmp(host1, last_host)) {
                strcpy(last_host, host1);
                if ((sin1.s_addr = inet_addr(host1)) == -1) {
                        if (host = gethostbyname_r(host1, &answer))
#ifdef POSIX
                                memmove(&sin1, host->h_addr, (sizeof sin1));    900
#else
                                        bcopy(host->h_addr, &sin1, (sizeof sin1));
#endif
                        else {
                                if (flags->debug_flag) {
                                        sprintf(bfr, "%s:  gethostbyname",
                                                host1);
                                        perror(bfr);
                                }
                                return(!strcmp(host1, host2));          910
                        }
                }
        }
        if ((sin2.s_addr = inet_addr(host2)) == -1) {
                if (host = gethostbyname_r(host2, &answer))
#ifdef POSIX
```

```
                memmove(&sin2, host->h_addr, (sizeof sin2));
#else
                        bcopy(host->h_addr, &sin2, (sizeof sin2));
#endif                                                                  920
                else {
                        if (flags->debug_flag) {
                                sprintf(bfr, "%s:  gethostbyname", host2);
                                perror(bfr);
                        }
                        return(!strcmp(host1, host2));
                }
        }
        return(!bcmp(&sin1, &sin2, (sizeof sin1)));
}                                                                       930

/*
 * Owner list code, originally written by jfc
 */
int clean_attachtab(atp, flags)
        struct _attachtab *atp;
        struct flags *flags;
{
        struct passwd *pw;
        int i;                                                          940
        for(i=0;i<atp->nowners;i++)
                if(NULL == (pw = getpwuid(atp->owners[i]))) {
                        int j;
                        /* Unmap */
                        if (atp->fs->type == TYPE_NFS && flags->do_nfsid) {
                                if (flags->debug_flag)
                                        fprintf(stderr,
                                                "nfs unmap(%s, %d)\n",
                                                atp->host, atp->owners[i]);
                                (void) nfsid(atp->host, atp->hostaddr[0],       950
                                        MOUNTPROC_KUIDUMAP, 1,
                                        atp->hesiodname, 0,
                                        atp->owners[i], flags);
                        }
                        for(j=i+1;j<atp->nowners;j++)
                                atp->owners[j-1] = atp->owners[j];
                        atp->nowners--;
                        i--;
                }
        return atp->nowners;                                            960
}

void add_an_owner(atp,uid)
      uid_t uid;
      struct _attachtab *atp;
{
        register int i;
        for(i=0;i<atp->nowners;i++)
                if(atp->owners[i] == uid)
                        return;                                         970
```

```
        if(atp->nowners < MAXOWNERS)
                atp->owners[atp->nowners++] = uid;
        else
          /* fail silently */;
}

int is_an_owner(at,uid)
        uid_t uid;
        struct _attachtab *at;
{                                                                    980
        register int i;
        if (!at->nowners)
                return(1);          /* If no one claims it, anyone can have it */
        for(i=0;i<at->nowners;i++)
                if(at->owners[i] == uid)
                        return 1;
        return 0;
}

#ifdef ZEPHYR                                                        990
int wants_to_subscribe(at, uid, zero_too)
        uid_t uid;
        struct _attachtab *at;
        int zero_too;               /* also true if root ? */
{
        register int i;

        for(i = 0;i < at->nowners;i++)
                if((zero_too && at->owners[i] == 0) || at->owners[i] == uid)
                        return 1;                                    1000
        return 0;
}
#endif

int del_an_owner(at,uid)
        uid_t uid;
        struct _attachtab *at;
{
        register int i;
        for(i=0;i<at->nowners;i++)                                   1010
                if(at->owners[i] == uid) {
                        --at->nowners;
                        for(;i<at->nowners;i++)
                                at->owners[i] = at->owners[i+1];
                        return at->nowners;
                }
        if(at->nowners == 0)
                at->owners[0] = -1;
        return at->nowners;
}                                                                    1020

char *ownerlist(atp)
        struct _attachtab *atp;
{
```

```
        static char ret[256];
        int i,len=1;
        if(atp->nowners == 0)
                return("{}");
        else if(atp->nowners == 1)
                return struid(atp->owners[0]);                          1030

        ret[0] = '{';
        for(i=0;i<atp->nowners;i++) {
                char *u = struid(atp->owners[i]);
                int tmp = strlen(u);
                if(i)
                        ret[len++] = ',';
                if(len+tmp >= 255) {
                        ret[len++] = '}';
                        ret[len] = '\0';                                1040
                        return ret;
                }
                strcpy(ret+len,u);
                len += tmp;
        }
        ret[len++] = '}';
        ret[len] = '\0';
        return ret;
}
                                                                        1050

int parse_username(s)
        const char *s;
{
        struct passwd   *pw;
        const char      *os = s;

        pw = getpwnam((char *)s);
        if (pw)
                return(pw->pw_uid);                                     1060
        else {
                if (*s == '#')
                        s++;
                if (isdigit(*s))
                        return(atoi(s));
                fprintf(stderr, "Can't parse username/uid string:  %s\n", os);
                exit(1);
                /* NOTREACHED */
        }
}                                                                       1070


int parse_groupname(s)
        const char *s;
{
        struct group *gr;

        gr = getgrnam((char *)s);
```

```
        if (gr)
                return(gr->gr_gid);
        else {
                if (*s == '#')
                        s++;
                if (isdigit(*s))
                        return(atoi(s));
                fprintf(stderr, "Can't parse groupname/gid string:  %s\n", s);
                exit(1);
                /* NOTREACHED */
        }
}
```

## A.3.21 zephyr.c

```
/*      Created by:     Robert French
 *
 *      $Source: /afs/dev.mit.edu/project/release/source/src/athena/bin/attach/RCS/zephyr.c,v $
 *      $Author: miki $
 *
 *      Copyright (c) 1988 by the Massachusetts Institute of Technology.
 */

#ifndef lint
static char rcsid_zephyr_c[] = "$Header:  /afs/dev.mit.edu/project/release/source/src/athena/bin/attach
#endif

#include "attach.h"
#ifdef ZEPHYR
#include <zephyr/zephyr.h>
#include <signal.h>

#ifdef _HIGHC_
#define min(x,y)        _min(x,y)
#else
#define min(x,y)        ((x)<(y)?(x):(y))
#endif

#define ZEPHYR_MAXONEPACKET 5
Code_t ZSubscribeTo(), ZUnsubscribeTo();

static ZSubscription_t subs[ZEPHYR_MAXSUBS];
static int num_subs = 0;

static pthread_mutex_t zephyr_lock = PTHREAD_MUTEX_INITIALIZER;

static int zephyr_op(func, flags)
    Code_t (*func)();
    struct flags *flags;
{
    static int inited = 0;
```

```
    static int wgport = 0;
    int count, count2;
    ZSubscription_t shortsubs[ZEPHYR_MAXONEPACKET];
    Code_t retval;                                                          40

    if (inited < 0)
        return 1;

    if (!num_subs)
        return 0;           /* Can't lose if doing nothing */

    if (!inited) {
        if ((retval = ZInitialize()) != ZERR_NONE) {
            com_err(config.progname, retval,                                50
                    "while intializing Zephyr library");
            inited = -1;
            return 1;
        }
        if ((wgport = ZGetWGPort()) == -1) {
            /*
             * Be quiet about windowgram lossage
             */
            inited = -1;
            return 1;                                                       60
        }
        inited = 1;
    }

    for (count=0; count<num_subs; count += ZEPHYR_MAXONEPACKET) {
        for (count2=0; count2<min(ZEPHYR_MAXONEPACKET,num_subs-count); count2++)
            shortsubs[count2] = subs[count+count2];
        if ((retval = (func)(shortsubs,
                             min(ZEPHYR_MAXONEPACKET,num_subs-count),
                             wgport)) != ZERR_NONE) {                       70
            fprintf(stderr, "Error while subscribing:  %s\n",
                    error_message(retval));
            inited = -1;
            return 1;
        }
    }
    return 0;
}

void zephyr_addsub(class, flags)                                            80
    char *class;
    struct flags *flags;
{
    pthread_mutex_lock(&zephyr_lock);

    if (num_subs == ZEPHYR_MAXSUBS) {
        pthread_mutex_unlock(&zephyr_lock);
        return;
    }
                                                                            90
```

```
    if (flags->debug_flag)
            printf("Subscribing to zephyr instance %s.\n", class);
    subs[num_subs].zsub_recipient = "*";
    subs[num_subs].zsub_classinst = strdup(class);
    subs[num_subs].zsub_class = ZEPHYR_CLASS;
    num_subs++;

    pthread_mutex_unlock(&zephyr_lock);
}
                                                                    100
int zephyr_sub(iszinit, flags)
int iszinit;
struct flags *flags;
{
    pthread_mutex_lock(&zephyr_lock);
    if(zephyr_op(ZSubscribeTo, flags) && iszinit)
      {
          flags->status = ERR_ZINITZLOSING;
          pthread_mutex_unlock(&zephyr_lock);
          return FAILURE;                                           110
      }
    pthread_mutex_unlock(&zephyr_lock);
    return SUCCESS;
}

int zephyr_unsub(iszinit, flags)
int iszinit;
struct flags *flags;
{
    pthread_mutex_lock(&zephyr_lock);                               120
    if(zephyr_op(ZUnsubscribeTo, flags) && iszinit)
      {
          flags->status = ERR_ZINITZLOSING;
          pthread_mutex_unlock(&zephyr_lock);
          return FAILURE;
      }
    return SUCCESS;
    pthread_mutex_unlock(&zephyr_lock);
}
#endif                                                              130
```

# Appendix B

# Notes on Currently Available Tools

Distributions of MIT Pthreads are available to the world at large via FTP from `sipb.mit.edu:pub/pthreads`. In the Athena environment, it is best used by using `/afs/sipb.mit.edu/project/pthreads/stable/bin/pgcc` as your compiler. `pgcc` invokes `gcc` with the necessary arguments to find the MIT Pthreads header files and libraries. Documentation for MIT Pthreads is available in hypertext form at `http://www.mit.edu:8001/people/proven/pthreads.html`.

While debugging a multithreaded program using MIT Pthreads, it is sometimes useful to examine the data structures internal to MIT Pthreads. In particular, you can walk down the list `pthread_link_list` to see what threads are doing; for a description of the the possible thread states, see `include/pthread/state.def`.

On systems with the `SIGINFO` signal, you can send a `SIGINFO` to a program using MIT Pthreads to get a dump of thread information.

The source for DCE threads is present in `/afs/dev.mit.edu/reference/dce`, but is not available to most users.

# Bibliography

[1] Dyer, Stephen P.. "The *Hesiod* Name Server." Available at `ftp://athena-dist.mit.edu/ftp/pub/ATHENA/usenix/hesiod.ps`

[2] Institute of Electrical and Electronics Engineers. *Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API)—Amendment 2: Threads Extension [C Language].* Draft 10. Institute of Electrical and Electronics Engineers. The status of POSIX standards is available at `http://wxweb.msu.edu/~crs/stds/pasc/standing/sd11.html`

[3] The Massachusetts Institute of Technology's Project Athena materials are available at `ftp://athena-dist.mit.edu:pub/ATHENA`

[4] Provenzano, Chris. "Pthreads." Available at `http://web.mit.edu/proven/www/pthreads.html`

[5] Raeburn, Ken. "A Common Error Description Library for UNIX." Available as `com_err.texinfo` in the `com_err` distribution at `ftp://athena-dist.mit.edu/ftp/pub/ATHENA/tools/com_err.tar.Z`

[6] Salz, Richard. "InterNetNews: Usenet Transport for Internet Sites." Available at `ftp://ftp.uu.net:/networking/news/nntp/inn/inn.usenix.ps.Z`

[7] Sun Microsystems. "Network Information Service Plus(NIS+): An Enterprice Naming Service." Available at `http://www.sun.com:80/sunsoft/solaris/networking/NIS-Admin-WP.html`

[8] Sun Microsystems. "pthreads and Solaris threads: A comparison of two user-level threads APIs." Early Access Edition, May 1994. Available at `http://www.sun.com/sunsoft/Developer-products/sig/threads/doc/pthreads_comparison.ps`

[9] Sun Microsystems. "Threads SIG Frequently Asked Questions." Part number 95114-001. Available at `http://www.sun.com/cgi-bin/show?sunsoft/Developer-products/sig/threads/faq.html`