+5

A Secure Distributed Printing System

by

Alexandra Ellwood

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 21, 1997

Copyright 1997 Alexandra Ellwood. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author_____

Department of Electrical Engineering and Computer Science

May 21, 1997

Certified by_____

James D. Bruce

Thesis Supervisor

Accepted by_____

Arthur C. Smith

Chairman, Department Committee on Graduate Theses

OCT 29 1997

A Secure Distributed Printing System

by

Alexandra Ellwood

Submitted to the

Department of Electrical Engineering and Computer Science

May 21, 1997

In Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This paper describes the motivation and design for a secure distributed printing system for MIT's Athena Computing Environment. Modern secure printing systems use encryption between clients and print servers. When the servers are connected to the printers by serial or parallel cables, the physical security of the cables protects the print jobs. Unfortunately, in the Athena environment, servers and printers are connected by ethernet networks, and because printers cannot encrypt, other machines on the networks a print job traverses can copy it. This paper proposes a solution to this problem: mini-supervisors. Mini-supervisors are inexpensive and easy to maintain devices which, when physically secured to printers, maintain the security of print jobs between servers and printers even when they are connected by a network. The paper then demonstrates this idea with a prototype using San Diego State University's LPRng printing system and provides an analysis of the prototype's performance.

Thesis Supervisor: James D. Bruce

Title: Vice President for Information Systems and Professor of Electrical Engineering

1. Introduction

While many of the distributed printing systems currently available claim some form of "security" [10,13,14,19], their use of the word secure can be misleading. Although most printing systems provide some form of access control to the printers, only a few actually protect data as it is sent to printers. Anything sent to a printer can be copied as the data travels through the networks which connect the client computer to the printer, usually without leaving any evidence that the information is no longer secret. If the printing system does not protect print jobs, other people on the networks the print job traverses can copy everything sent to the printer and view or print out copies for themselves.

In the past, separate mechanisms to provide access control and secrecy when printing were unnecessary. Print jobs were sent from large mainframes located in a central data center to fast high-end printers in the same area. Access control was provided by limiting who could log into the mainframe, and print job data remained secret because it traversed a small mainframe network to which only system administrators had access. However, with modern distributed systems, print jobs are submitted by client machines to print servers which may be far away from both the client computer and the printer. Now print job data may traverse large networks before getting to the printer, and the group of people who can get access to this data is no longer limited to system administrators [11].

In response to this problem, departments within the Massachusetts Institute of Technology have asked MIT's Information Systems to provide a secure networked printing system, so that documents containing confidential data such as grade reports can be printed to printers on MITnet. As my thesis, I have designed and built a prototype of such a system by modifying an existing version of the Line Printer Daemon (LPD) networked printing system.

2. Background

In order to identify why printing systems are insecure, I have analyzed the currently available printing system models which provide some form of access control or secrecy.

Because these systems can have very different goals, they can be very hard to compare. I have attempted to compare them solely based on their ability to control access to the printer and maintain the secrecy of the data sent to it.

2.1 Variations on LPD: Athena LPD, Net-Print, SAPLPD and LPRng

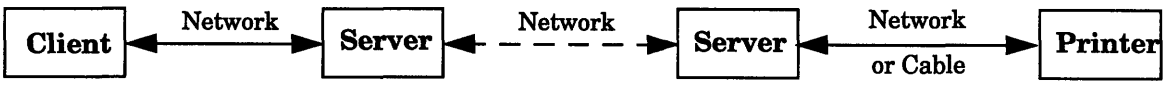

Figure 1: LPD Client-Server Arrangement

One of the oldest distributed printing systems is U.C. Berkeley's Line Printer Daemon (LPD) protocol. The overall structure of the system is shown in Figure 1. When a user prints a document to a printer, the client looks up the printer's name in a "printcap" file on the client to get the name of the server and the name of a print queue on that server to send to. The client then sends that server the name of the print queue the client wants the job to be printed to, the print job data, and a control file. The control file identifies the name of the print job, user and client, as well as any printing options the client might wish to specify (such as the number of copies to print). For each print job, the server writes the print job data and associated control file into a queue directory, and then sends it either to another LPD server or to a printer, depending on how the printing system has been configured. In this way, print data is sent from the client through one or more servers to the printer [12]. This flexible design allows system administrators to organize their servers in many ways. Unfortunately, the original implementation of LPD provided minimal access control[1] and no secrecy. A client could fill in any user name in the control file it sent (thereby becoming any user), and the print job traveled in the clear across the network between the client, servers and printer.

In order to better regulate who can use their printers, MIT's Project Athena added Kerberos v4 authentication to the LPD protocol. Athena LPD uses the user's Kerberos v4 identity to determine who is attempting to use a printer. If the user is not authorized to use a

1. Access control was provided by only allowing connections from a privileged port. Because this required clients be setuid root, they could only have been created by root and could be assumed to report the actual name of the user when communicating with the server.

printer, the server refuses the connection. If the printer is directly connected to the network, clients are prevented from circumventing the server and talking directly to the printer because the printer is configured to only accept connections from the server's IP address[2]. Although Athena LPD does not provide secrecy for print jobs, adding encryption between the clients and servers and between the servers using Kerberos v4 session keys is possible.

Cornell University's Net-Print also uses the LPD protocol for printer communication. Like Athena LPD, it only provides access control. However, it is worth analyzing because of the modular way it handles authenticating the user. First, the client sends an unauthenticated print job request to the server. The LPD server saves the client's IP address and uses it to call back to a separate process on the client called a "sidecar." The LPD server and the sidecar use Kerberos v4 authentication to verify that the user who claimed to have submitted the request actually did [9][3].

Net-Print's "sidecar" design can be implemented with any protocol that needs authentication. The design allows administrators to use vendor clients without having to modify them to include their own authentication protocol. Although they still have to modify the server software, the server only needs to run on a single operating system. Client software often must run under Windows, MacOS and many Unix platforms and maintaining that software comprises most of the development and support effort in a client-server system. Unfortunately, because of the sidecar implementation, Kerberos session keys cannot be used to encrypt communication between the client and server and between the server and other servers. This limitation prevents Net-Print from easily implementing secrecy.

---

2. IP spoofing can overcome this restriction. However, Athena LPD's maintainers do not consider it a problem because the method is not used very often. Also, the current authentication system only prevents non-MIT people from printing, so circumventing the printing system is not very useful.
3. With the information available on Net-Print, I could not determine if the server uses IP addresses as the sole way of identifying clients. If so, a user could use IP spoofing to forge print requests from another user who is also logged in. By forging a print job as the second user and faking his client's IP address, the client can convince the server to talk to the second user's sidecar, which will respond that the user is in fact logged in on that machine, authenticating the print job. This can be fixed in a number of ways. One way is to have the sidecar make a simple one-way hash of the print job before it is sent by the client. The server can then verify that the sidecar it is authenticating the same client who made the request by comparing its hash to the one the sidecar has.

SAP AG's SAPLPD and San Diego State University's LPRng both implement secrecy in addition to access control. SAPLPD uses the Generic Security Services API (GSS-API) to negotiate access control and secrecy protocols between the client and server and between servers. Currently SAPLPD negotiates either Kerberos v5 or GMD's Security Development Environment (SECUDE) [14]. LPRng uses either Kerberos v5 or Pretty Good Privacy (PGP) for authentication and encryption. LPRng clients and servers negotiate between the two possible security protocols using a extension to the LPR protocol proposed by Professor Patrick Powell, author and maintainer of LPRng.

Unfortunately none of the four LPD variants implement encryption between the server and its printers. System administrators who want security must connect servers and printers with a serial or parallel cable so that at the link between the server and the printer, the print job does not traverse the network. Unfortunately, this requires each printer or group of printers in a room have its own server, increasing the number of system administrators required to maintain the servers. This also requires that the server be located physically close to the printer, which increases support costs because physical access to the server must be restricted.

2.2 The Document Printing Application Specification

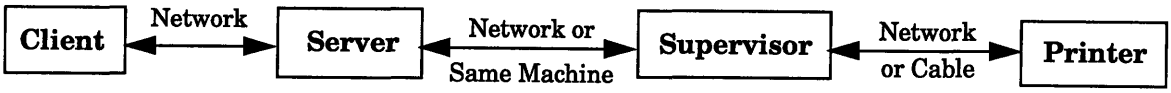

Figure 2: The Palladium Client-Server-Supervisor Model

In the late 1980s, MIT's Project Athena created a new distributed printing model called the Document Printing Application specification (DPA), which defines the attributes and objects a distributed printing system should support [10]. In cooperation with Digital, IBM and HP, Project Athena produced an experimental implementation of a DPA-based system called Palladium. Palladium consists of a client, a server and a supervisor organized as shown in Figure 2. It supports numerous features, including the ability to determine the capa-

bilities of the printer from the client and the ability to have a print queue correspond to more than one printer. Palladium also provides access control through Kerberos v4, but does not provide secrecy of print jobs [7]. Because it is a prototype, Palladium is not suitable for use in the Athena Computing environment, and Project Athena switched to the system they currently use: Athena LPD. Despite Palladium's failure in a production environment, a number of vendors have implemented their own printing systems based on the DPA specification and the Palladium design. Among these are Xerox's Printxchange, IBM's Printing Systems Manager (PSM), and Novell's Distributed Print Service for NetWare 4.1. Because they all use the DPA specification, components of each system have been developed to communicate with one another [10,13,19].

Printxchange and PSM are very similar with respect to security. Both use the Open Software Foundation's Distributed Computing Environment (DCE) security services to implement access control [10,19]. Although neither system provides secrecy, encryption can be easily implemented between the client and server and between the server and supervisor using the existing DCE authentication system. But as with LPD-based systems, the supervisor and the printer must be connected with a secure cable to provide complete secrecy. The benefit of a Palladium-based system is that the supervisor is a much less complicated daemon than the server and requires a less powerful machine. This reduces the cost of the equipment which needs to be placed next to each printer.

Novell's Distributed Printing Services provides access control through Novell's NetWare Directory Services (NDS) and NetWare's Security Management Services. Physical printers are represented by servers called Virtual Printers (VPs), which register themselves with NDS. Clients must then authenticate to NDS using NetWare security services to be able to print to the VPs [13]. As with Printxchange and PSM, Distributed Printing services does not provide secrecy, however communication between clients and VP servers can be encrypted as an extension of the existing authentication system

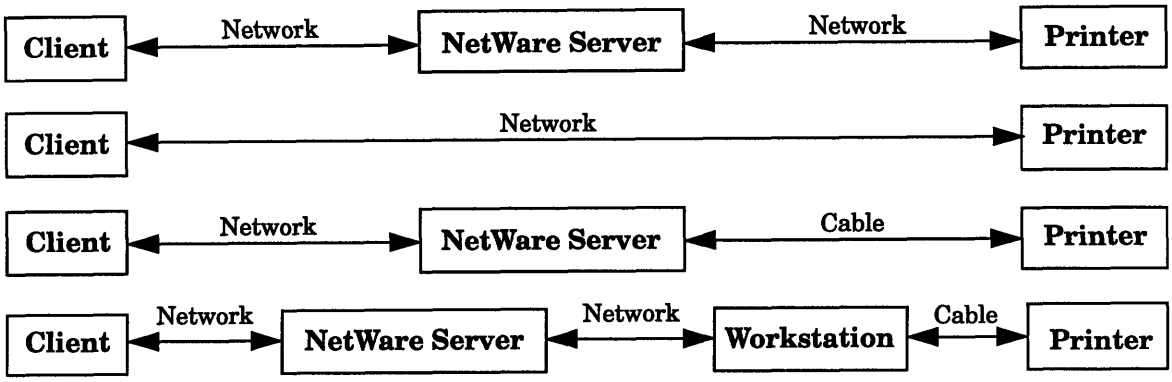

Figure 3: Distributed Printing Services Printer Configurations

Distributed Printing Services provides the services outlined in the DPA specification, but does not follow the Palladium client-server-supervisor model. Instead it provides four different configurations for where the printer can reside on the network. These are shown in Figure 3.

In the first configuration, the printer is directly connected to the network. A NetWare server running a VP handles authentication and print jobs. As in the other printing systems, this configuration cannot fully provide encryption because the connection between the printer and the NetWare server is insecure. In the second configuration, the printer is also directly connected to the network, but it represents itself as a VP. This configuration requires that the printer have an "embedded VP server". If this VP server is modified to support encryption, the printer itself could provide direct secrecy. In the third configuration, the printer is connected to a NetWare server running a VP via a secure serial or parallel cable [13]. This arrangement will provide secrecy if the system provides encryption from the client to the VP server. Finally, the printer can be attached to a workstation on the network via a secure cable. The VP server runs on a NetWare server and sends print jobs to the workstation to be forwarded to the printer. This arrangement can provide encryption all the way to the workstation. However, if the users of the workstation are not trusted, the system cannot be secure because those users have physical access to the machine which decrypts the print job.

2.3 CUPID and the Print-On-Demand Model

The Print-On-Demand model takes a much broader view of printing. Motivated by the desire of publishers to offset printing costs, Print-On-Demand suggests a "distribute, then print" model rather than the current "print, then distribute" model of publishing [17]. In other words, rather than purchasing a physical book, customers will now purchase an electronic copy of that book through an on-line service and print it out on their own site's printing system. This system can help schools, which currently photocopy hundreds of pages of articles and textbook chapters for class notes, often in violation of copyright laws. This method of purchasing can allow them to legally obtain the portions of books they need in an electronic format, making class notes less expensive and less time consuming to produce.

The Consortium for University Printing and Information Distribution (CUPID) has taken on the task of designing a distributed networked Print-On-Demand system which protects the interests of both publishers and customers [4, 17]. This system is currently only in the design phase and a detailed examination of its security capabilities is beyond the scope of this paper. However, it is important to note that for a Print-On-Demand system to protect the interests of the publisher, it must provide access control as a way of identifying customers and secrecy to keep people from copying publications as they are transmitted to the customer's printer.

3. Design Requirements

In order to design a secure distributed printing system which meets the needs of the Athena Computing Environment, I approached the Athena Server Operations (ASO) group (the people who maintain the current Athena print services) to find out what their support requirements and the requirements of their users are. They identified the following four major requirements:

3.1 Location of Print Servers

Their first requirement is that they need to be able to place the system's servers in their machine rooms, which are often very distant from the location of the physical printers. This is currently the main reason that the Athena Computing environment does not provide secrecy through one of the existing secure printing systems. In order to implement secrecy in any of the existing systems, ASO must dedicate a server to each printer or group of printers. The servers must be physically close to the printers to allow the printers to be connected to the server by serial or parallel cables. In addition, the server and cables need to be protected from tampering.

For a large computing environment like MITnet with many printers, providing a server for each group of printers is too expensive. Staff at Information Systems estimate that each server would cost approximately $4000 and that maintenance and protection of access to the server would be about $1000 per year [11]. In addition, supporting these servers would be unreasonably time consuming for ASO staff because there would be a large number of servers located all over campus.

3.2 A Familiar System

Their second requirement is that the system be familiar. Currently the ASO group is understaffed, and although they are trying to fill their open positions, the new hires will still need to be trained. As a result, over the next year they will not have the resources to learn how to set up and maintain a completely new printing system.

There are currently two printing systems in use in the Athena Computing environment: Athena LPD and SAPLPD. Because all LPD systems provide very similar interfaces, my system can meet this requirement by building off one of the existing LPD implementations. This is ideal, because two such systems (SAPLPD and LPRng) already provide authentication and encryption between clients and servers and between servers and servers.

3.3 An Inexpensive System

Their third requirement is that the system be inexpensive. Because there is currently only a small demand for secure printing, Information Systems cannot justify spending a lot on a secure printing system. In addition, if my design is too expensive, they might as well use one of the existing printing systems which provide partial security (such as SAPLPD) and dedicate a server to each of the printers which need to be secure.

3.4 A Printer Independent System

Their final requirement is that the system must not rely on a particular printer model or manufacturer. The departments which have requested a secure printing system already own and use many different kinds of printers. Some of these printers shrink-wrap documents or print them into locked bins for access control, and no single vendor provides printers with every ability that any department might want. As a result, Information Systems cannot expect every department to switch to one vendor or one model of printer. In addition, ASO staff do not want to be constrained to one vendor or one printer model, because if the vendor stops providing acceptable printers, they want to be able to quickly turn to other vendors.

4. Possible Designs

Since my system can satisfy ASO's second requirement by building off an existing secure distributed printing system, my design can avoid specifying the entire printing system and instead focus on the weak link in all existing secure printing systems: communication between the server and the printer. All existing printing systems require that this communication be performed over a secure serial or parallel cable to implement secrecy. This is because while end-to-end encryption between clients and servers is not difficult, most printer operating systems and hardware do not have the full capabilities of a workstation or server[4], mak-

4. Namely, most printers do not have operating system features such as memory and file system protection or a means of permanent storage (such as a hard disk). Although some of these features can be added, they are usually expensive. Examples of specific capabilities which PostScript printers lack are discussed in section 4.2.

ing end-to-end encryption between servers and printers directly connected to a network much more difficult.

An additional benefit of concentrating only on the communication between server and printer is that my design may be applied to almost any distributed printing system available. All of the existing printing systems have a server or supervisor which talks directly to printers on the network. By using my design to add secrecy to this link and adding some form of end-to-end encryption between the clients, servers and supervisors in the system, most of the existing secure printing systems can be made fully secure in all configurations. This is especially valuable because each of the existing secure printing systems has different strengths and weaknesses. If all the available secure printing systems are made fully secure, then administrators can choose the system which best suits their environment without having to sacrifice secrecy.

In trying to add end-to-end encryption between print servers and printers, I considered three possible designs which, to some degree, meet the requirements of the Athena Computing Environment.

4.1 Support from Printer Manufacturers

One simple solution is to convince vendors to add standardized end-to-end encryption abilities to their printers, either in the printer's networking hardware or in the printer's operating system. Unfortunately, because there is not a high demand for secure printing in most businesses, only the vendors of high-end secure printers (such as those which shrink-wrap or print to locked bins) would be interested in a printer security standard. As a result, this solution would become dependent on particular printer vendors or models, and the system would not longer satisfy ASO's fourth requirement.

4.2 Encryption in PostScript

A second possible solution is to implement end-to-end encryption abilities as a filter procedure written in Adobe's Level 2 PostScript language and store it on a ROM card or hard

drive in the printer. Because PostScript is a programming language, PostScript print jobs are actually programs which tell the printer how to draw each page. However, PostScript procedures can also perform mathematical calculations such as encryption. By calling a PostScript decryption procedure stored on the printer's hard drive or ROM card, a print job can effectively decrypt itself.

Most printers come with an option for Level 2 PostScript, and many of these also support ROM cards or hard drives to permanently store fonts and filters. Although printers which cannot store filters cannot be used securely, most of these printers are low-end desktop models which would be inappropriate for a networked printing system.

In order to securely send a PostScript document to the printer, a server sends a job in the following format:

$$Execute\_Filter\_Job([Ks]^{Km}, [Check\_Page\_Count\_Job(Page\_Count, Job)]^{Ks})$$

Km is a master key shared between the server and printer (a public key model could also be designed). *Ks* is a random session key generated by the server for every job. *Job* is the PostScript job submitted by the user, and *Page_Count* is the printer's current page count. The current page count is a monotonically increasing number stored on the printer. The page count cannot be changed, and only resets when the variable it is stored in overflows.

To generate the secure PostScript job, the server makes a PostScript query to the printer for the current page count in clear text and stores it in *Page_Count*. The server then generates a PostScript job (represented as *Check_Page_Count_Job*) which checks the page count on the printer, and if it is the same as *Page_Count*, executes *Job*. The server encrypts *Check_Page_Count_Job* in *Ks*, and creates a PostScript job (represented as *Execute_Filter_Job*) which passes $[Ks]^{Km}$ and $[Check\_Page\_Count\_Job]^{Ks}$ to the PostScript filter on the printer. The server then sends *Execute_Filter_Job* to the printer. Once the job gets to the printer, the PostScript filter decrypts *Ks* with its own copy of *Km*, decrypts *Check_Page_Count_Job* with *Ks*, and executes *Check_Page_Count_Job*.

The reason for using a session key (rather than the master key) to encrypt the job is to

prevent a malicious user from obtaining large amounts of known text encrypted with the master key by sending specially chosen jobs to the printer and looking at the encrypted versions of those jobs. Since every job is encrypted with a different session key, a malicious user has only one job worth of known text to attack the session key. Since the session key is random data, the attacker will have to discover the session key to check any guesses at the master key.

The reason for checking the page count is to prevent malicious attackers from packet sniffing the encrypted job and printing it out on the same printer later. This kind of attack is referred to as a "replay attack." Since the job checks the page count, the printer will only print the job when the page count matches the actual page count stored on the printer. Assuming every job will increment the page count at least by one (by printing at least one page), a document cannot be printed twice with the same encrypted job. Although the page count will eventually roll over, an attacker would have to monitor the printer constantly to be able to submit a job exactly when the page count was the same value.

The most difficult task in designing this system is protecting the master key on the printer. The PostScript language was written for flexibility rather than security, so keeping jobs from printing out the key or overwriting the decrypting filter is very difficult. However, by using a special version of the *bind* operator, the *executeonly* operator and the *noaccess* operator, a PostScript filter can be placed on the printer such that all its variables are non-accessible and cannot be read. By installing this filter outside the printer's server loop (where the initial state of the printer can be modified), and then starting the server loop, the filter will remain protected as long as users cannot exit the server loop with *exitserver* operator.

There are several problems with designing a system like this. First, the system is only as secure as the server loop password, which must be provided as an argument to the *exitserver* operator to exit the server loop. Because the length of the server loop password is defined by vendors, only printers with reasonably long server loop passwords will be secure. In addition, some printers have easily accessible button combinations which if held down while the printer is booting, reset the server loop password to a known default. This limits the

printers which the system can be run on, making the system somewhat printer dependent.

A second problem is that printer hard drives are expensive and customized ROM cards require cooperation from the vendors. Because printer hard drives use proprietary device interfaces, a 500 MB hard drive can cost over $1000. ROM cards may also be expensive, and because each printer takes a different kind of ROM card, custom ROM cards must be ordered from the printer's manufacturer.

Finally, the security of the key stored on the printer depends on the PostScript operators behaving as they are specified in the Adobe PostScript Reference Manual [1]. However, many modern printers come with PostScript "clones" developed by other software companies. Sometimes these implementations of the PostScript specification do not behave correctly, possibly compromising the security of the system. Limiting the system to only Adobe PostScript printers would also make the system too printer-dependent.

While encrypting in PostScript solves the end-to-end encryption problem, the system is complex, expensive and too dependent on the implementation of PostScript. I approached ASO with this design and they rejected it based on its printer dependence and the possibly high cost of the implementation.

4.3 Mini-Supervisors

After trying to implement end-to-end encryption inside the printer using a PostScript procedure, I realized that printer operating systems are not secure enough to decrypt their own jobs without additional vendor support. As a result, my final design draws off Palladium's concept of printer supervisors. While placing an entire $4000 server next to each printer is too costly, the functionality of an entire server is not necessary to merely decrypt incoming jobs. By placing the minimum amount of hardware needed to decrypt print jobs in an easy to secure and maintain box, I can build an inexpensive device which acts as a miniature supervisor. This machine will reside on the network as if it is the printer, decrypting print jobs and forwarding them through the printer's serial or parallel port. Since almost all printers come with a serial or parallel interface, this device will be virtually printer independent.

As an initial implementation, the "mini-supervisors" and servers will share secret keys. Each printer and mini-supervisor will have an associated secret key stored in ROMs on the mini-supervisor and on local disk on the server. This secret key would be used to negotiate session keys for encryption between the server and mini-supervisor.

However, in the secret key arrangement, the machine talking directly to the mini-supervisor must know the secret key. As a result, only trusted machines may communicate directly with the mini-supervisor using encryption. In some cases, system administrators may want to provide encrypted printing services for untrusted clients and servers which wish to communicate directly with the printer rather than through a maintained server. A solution to this problem would be to use a public key system in which the mini-supervisor and printer have an associated key pair. The private key would be stored inside the mini-supervisor in ROMs. The public key would either be stored on the print server to limit encryption services to the server's printing system or made publicly available through an online service such as the Athena Computing Environment's Hesiod database. Having the public key on an online service would allow multiple untrusted printing systems to use the encryption services of the mini-supervisor without being able to decrypt jobs from other servers.

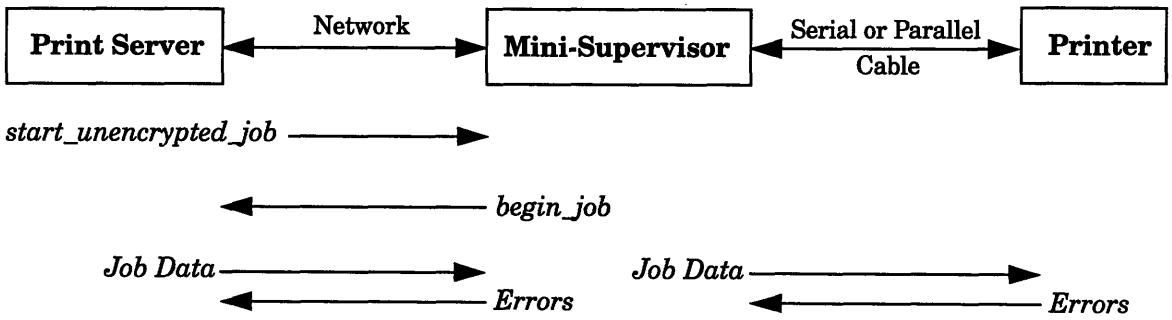

Figure 4: Sending an Unencrypted Job

Because encryption requires additional computation, encrypting and decrypting print jobs will undoubtedly slow down the processing of a job. Data which does not need to be kept secret should not have to be encrypted. As a result, the mini-supervisor accepts two kinds of job connections: an unencrypted job connection and an encrypted job connection. As shown in

Figure 4, to print an unencrypted job, the server sends a *start_unencrypted_job* message. The supervisor will either respond with a *busy* message, indicating that the server should try again later, or with a *begin_job* message. If the server receives the *begin_job* message, then it may begin treating the mini-supervisor as a printer. The mini-supervisor passes messages directly from the server to the printer and vice-versa. By sending job data and listening to the replies, the server can print out the job and handle errors. When the server is done communicating with the printer, it closes its connection to the mini-supervisor and the mini-supervisor waits for the next connection from the server.

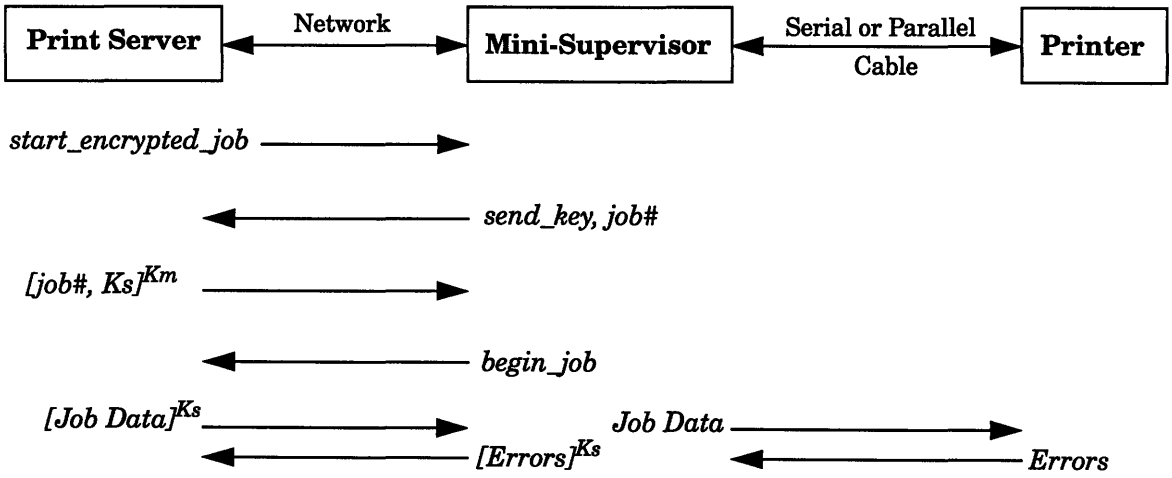

Figure 5: Sending an Encrypted Job

As shown in Figure 5, in order to print an encrypted job, the server sends a *start_encrypted_job* message. The mini-supervisor may respond either with a *busy* message, indicating that the server should retry later, or it may respond with a *send_key* message and a *job#*. The *job#* is a monotonically increasing counter maintained by the mini-supervisor which protects print job data from replay attacks. The job number may either be the page count of the printer (requiring that the mini-supervisor know how to query the printer for its page count), or a number stored by the mini-supervisor (requiring that the mini-supervisor have a small amount of non-volatile storage). Once the server receives the job number, it generates a session key *Ks* and sends $[job\#, Ks]^{Km}$, where *Km* is the secret "master" key stored by both the

mini-supervisor and server. This associates a session key *Ks* with the job number *job#*. The mini-supervisor decrypts the server's message with its private key and verifies that its current job number is *job#*. This prevents malicious users from replaying the session key message to print the same job twice.

If the job numbers match, then the mini-supervisor will send the server a *begin_job* message (otherwise it will send an error message). As in the unencrypted case, after this message has been sent, everything the server sends to the mini-supervisor will be forwarded directly to the printer and error messages from the printer will be forwarded back to the server. However, in the encrypted job case, all the messages sent by the server are encrypted with the session key and are decrypted by the mini-supervisor before being forwarded to the printer. Similarly, the mini-supervisor encrypts all error messages from the printer before forwarding them to the server. In this way all communication between the server and printer is encrypted. When the printer is done printing the job, it closes the connection. The mini-supervisor will then destroy *Ks* and increment the job number. If the job number is the printer's page count it will be incremented automatically by the printer.

The mini-supervisor will be easy to maintain because it is small, easy to lock down, and has no interface with which users can change its settings. Although users can unplug the power to the mini-supervisor, this only causes the same denial of service attack as unplugging the printer itself. The mini-supervisor can also be secured directly to the printer's serial or parallel port so that users cannot connect their own computers between the mini-supervisor and the printer (to copy print jobs after they are decrypted).

The mini-supervisor can be inexpensive because it only requires a low-end processor, a network card, a serial interface, a parallel interface, a small amount of RAM and some ROMs. Hewlett-Packard Corporation's JetDirect EX Plus and Digi International's FastPort 3400X provide the functionality of the unencrypted mode of the mini-supervisor at costs of $350 and $400 respectively [5,8]. This suggests that the cost of a mini-supervisor should be under $500 if a sufficiently large number of mini-supervisors are produced.

One concern users might have about this system is that the encryption between the client and server is decoupled from the encryption between the server and the mini-supervisor. The user's job is decrypted on the server and then re-encrypted when it is sent to the mini-supervisor. Anyone who has access to the server can read the user's jobs. However, this cannot be avoided for two reasons. First, the authors of existing printing systems should not need to replace their current authentication and encryption protocols to support mini-supervisors. In order to have encryption from the client to the mini-supervisor, both the clients and the servers of a printing system would have to be modified to support the mini-supervisor's encryption protocol. Second, most printing systems support some sort of job filtering on the server so that administrators can add features like enabling duplex printing and refusing jobs which will harm the printers. These filters must reside on the server to ensure that users cannot circumvent them by modifying their clients. However, in order for the filters to work, the server must have an unencrypted copy of the print job. As a result, decrypting the job on the server is a reasonable trade-off between security and maintainability.

The mini-supervisor design allows print servers to reside in machine rooms elsewhere on the network, can be used with any printing system, will be relatively inexpensive to build, and is independent of the printer's model and manufacturer. I approached Athena Server Operations with this design and they accepted it as one of the best possible solutions to the need for secure printing.

5. Implementation

To demonstrate the concept of mini-supervisors, I implemented a prototype mini-supervisor on a AST Premium II 386 SX/20 running the NetBSD v1.2 operating system. I chose a slow processor to show that the mini-supervisor does not require a $4,000 print server and therefore will not be too expensive to build. I selected the NetBSD operating system because I am familiar with Unix network programming and the NetBSD environment.

In order to simulate use in the Athena Computing Environment, the prototype pro-

vides printing service to the most common type of Athena printer: the Hewlett-Packard LaserJet 4si. The HP 4si printer is connected to the mini-supervisor by a IEEE 1284 compliant parallel connection. Unfortunately, although both the 386 SX/20 and the printer have hardware support for IEEE 1284 Extended Capabilities Parallel (ECP) mode, NetBSD v1.2 only has driver support for Compatibility Mode (sometimes referred to as "Centronics Mode")[5]. Compatibility Mode is significantly slower than ECP mode and provides only uni-directional communication with the printer (ECP mode is bi-directional) [16]. Implementing ECP mode drivers for NetBSD would improve the performance of the mini-supervisor and allow the print server to obtain status information from the printer.

In order to meet Athena Server Operations' requirement that the printing system be familiar, I chose San Diego State University's LPRng 3.2.1 running on a SPARCstation 4 as the printing system which communicates with the mini-supervisor. In addition to being familiar to the ASO group because it is a variant of Berkeley LPD, LPRng already provides authentication and encryption negotiation between the clients and the server using Professor Patrick Powell's proposed extensions to the LPD protocol (RFC 1179). Also, LPRng has freely available source code, making it easier to modify to communicate with the mini-supervisor.

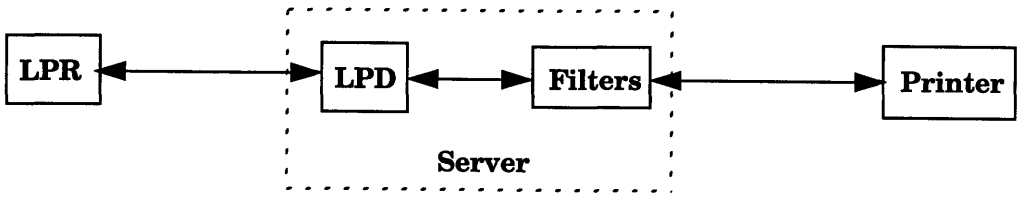

Figure 6: The LPRng Printing System

Figure 6 shows the overall structure of the LPRng printing system. LPR is the LPRng client, which negotiates Kerberos v5 authentication and encryption to open a secure connection with the LPRng LPD server. The LPR client then specifies one of the server's print queues to print to and sends a control file describing the print job and the job data file. The

5. The only operating system which currently has built-in support for IEEE 1284 ECP mode is Windows 95 [16]. I chose not to use Windows 95 because I was not familiar with it, and because I do not consider it an acceptable operating system for the mini-supervisor prototype.

LPD server then opens a connection to the printer associated with that print queue and passes this connection to a "print filter" program. The LPD server sends the job data file to the print filter and the filter translates the job data into a language (such as PostScript or PCL) which the printer understands and sends it to the printer. The filter also listens for any errors the printer might report and responds accordingly. As a result, the server does not need to know how to format jobs for every kind of printer, but instead uses a print filter specific to that type of printer. Servers may forward the print job through multiple print filters to enable features like adding a banner page to the front of each print job and checking the printer's page count to enforce print quotas.

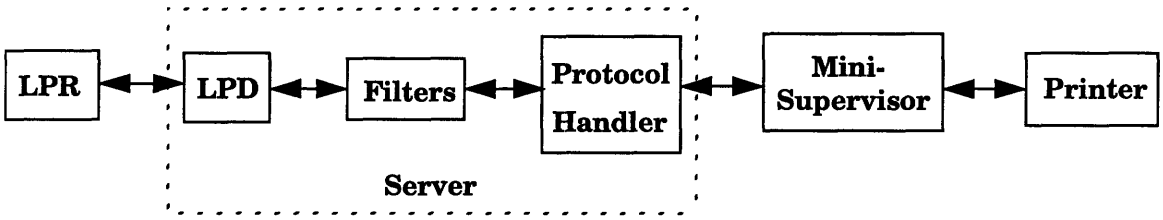

Figure 7: The Mini-Supervisor Printing System

To keep this layer of abstraction, I added an additional print filter called the "protocol handler." Figure 7 shows a block diagram of this design. The protocol handler acts as a filter between the normal print filters and the mini-supervisor, establishing session keys with the mini-supervisor, encrypting print job data and decrypting replies. The protocol handler and mini-supervisor effectively add an extra protocol layer on top of normal communication between the filters and the printer. Because the print job data still gets passed through the normal print filters, the protocol handler and the mini-supervisor do not need to know what languages the printer understands.

The protocol handler and the mini-supervisor encrypt print job data and responses from the printer using a fast Data Encryption Standard (DES) implementation written by Eric Young running in triple DES in cipher block chaining mode (CBC). Triple DES CBC mode uses three 64 bit keys and is sufficiently difficult to crack that it is considered adequate

for modern software systems [15]. Although DES is computationally intensive and will slow down the prototype mini-supervisor significantly, a production implementation of the mini-supervisor could contain a DES chip, which encrypts data very quickly in hardware.

6. Performance

In order to evaluate the performance of the mini-supervisor system, I compared the performance of LPRng in three different configurations: printing directly to the printer over the network, printing through the mini-supervisor in unencrypted mode, and printing through the mini-supervisor in encrypted mode. As a measurement of running time for print jobs, I used a stopwatch to measure the time from when the printer prints the job's banner page to when the printer finishes printing the last page of the job. The measurement of the time between the banner page and the last page accounts for the differences between the three models due to the introduction of the protocol handler, the mini-supervisor, Triple-DES encryption on the server and mini-supervisor, and the parallel cable connection between the mini-supervisor and printer. While this measurement does not look at the potential increased latency and decreased throughput due to encryption between the client and the server, this performance difference is specific to LPRng and is not a property of my prototype.

In order to show the performance of the different kinds of jobs a PostScript printer might receive, I measured the printing time for four different PostScript documents: Bible.ps, Thesis.ps, Graphics.ps and Bitmaps.ps. These documents reflect the most common type of print jobs which PostScript printers receive. I also made the four documents as large as possible (without using too much paper) to reduce the effects of noise from human error.

The first test job, Bible.ps, was constructed by prepending the King James version of the Bible in PostScript comments to a 104 character PostScript job which prints out a single page[6]. The test case prints out a page at the end of the job so I can tell when the printer has

6. I used the Bible because it is a large body of text which is easily available online. Any large body of text would have sufficed for this test case.

finished receiving data. Although no useful PostScript jobs have the characteristics of Bible.ps, this test case provides a good measure of how fast each of the test configurations can send data to the printer and a worst-case scenario. Because the text of the Bible is in PostScript comments, the printer performs no processing on all but the last 104 chars of the data. Therefore, the measurements for Bible.ps reflect the performance differences between the test configurations and are not inflated by the time the printer spends imaging and printing pages.

The other three test print jobs show what the average performance of the system would be. Bitmaps.ps is 20 pages of full page black and white bitmap images at 72 dots per inch. Graphics.ps is 20 pages of the same full page images as Bitmaps.ps, except that the images are generated using PostScript drawing routines. Thesis.ps is this thesis document (with appendices) and shows the printing times of an average text document with a few figures. These three jobs show the performance for the different kinds of images a PostScript printer actually receives: bitmap graphics, PostScript graphics, and formatted text.

File Name	Size (in bytes)	Networked Running Time	Unencrypted w/Mini-Supervisor Running Time	Encrypted Running Time
Bible.ps	4642470	69 sec	1193 sec (17.3x slower)	1769 sec (1.5 x slower)
Bitmaps.ps	3190884	73	807 (11.0)	1193 (1.5)
Graphics.ps	1011195	107	257 (2.4)	377 (1.5)
Thesis.ps	489940	479	484 (1.0)	491 (1.0)

Table 1: Printing Times for Normal and Prototype Printing System Configurations

Table 1 shows the timing measurements for the four different print jobs. The number in parentheses to the right of each of the unencrypted running times is the ratio of the unencrypted running time over the networked running time. Likewise, the numbers in parentheses to the right of each of the encrypted running times is the ratio of the encrypted running time over the unencrypted running time. These ratios show the performance differences

between the test configurations independent of the print job's size.

The results for Bible.ps show the actual data transfer speeds for each of the test configurations. The addition of the protocol handler, mini-supervisor, and parallel connection makes the transfer of the print job data 17.3 times slower than in the networked configuration. Fortunately, the other three test cases also show that the time the printer spends imaging and moving the physical pieces of paper masks this performance hit. As a result, print jobs which use fewer bytes of PostScript data per page will print faster because the page printing time masks more of the data transfer time. This is supported by the observation that Bitmaps.ps, which has approximately 400,000 bytes of data per page (one byte for every pixel) shows a larger performance hit than Thesis.ps, which only has approximately 2,500 bytes per page (one byte for every character).

One possible cause of the large performance hit which the mini-supervisor system incurs is the slow NetBSD parallel port driver. To test this hypothesis, I modified the mini-supervisor to print to a file on local disk rather than sending the file to the printer. I then reran all the tests measuring the time the mini-supervisor takes to write the file in order to see if the absence of the parallel cable significantly changed the results.

File Name	Size (in bytes)	Unencrypted Running Time	Encrypted Running Time
Bible.ps	4642470	20 sec	655 sec (32.7x slower)
Bitmaps.ps	3190884	15	449 (29.9)
Graphics.ps	1011195	5	147 (29.4)
Thesis.ps	489940	4	72 (18.0)

Table 2: Running Times for Printing System Prototype without Parallel Cable

Table 2 shows the time the mini-supervisor took to write each of the four print jobs to local disk in unencrypted and encrypted modes. The unencrypted mode results indicate that the parallel port is in fact causing most of the slowdown in the unencrypted configuration. A

faster parallel connection such as a IEEE 1284 ECP parallel connection would dramatically improve the unencrypted performance for the prototype and should be part of any production implementation.

However, the encrypted mode results show that the slower parallel port is also masking a massive performance hit due to the encryption between the server and the mini-supervisor. In order to verify that this is because of the slow processor on the mini-supervisor (and not the SPARCstation server), I ran the tests a third time, writing to a file on the mini-supervisor's local disk and only encrypting on the server. By turning off decryption on the mini-supervisor, I can see what fraction of the slowdown is as a result of the server.

File Name	Size (in bytes)	Unencrypted Running Time	Partial Encryption Running Time
Bible.ps	4642470	20 sec	40 sec (2x slower)
Bitmaps.ps	3190884	15	28 (1.9)
Graphics.ps	1011195	5	9 (1.8)
Thesis.ps	489940	4	6 (1.5)

Table 3: Running Times for the Prototype without Parallel Cable and Partial Encryption

Table 3 shows the running time for the system when decryption on the mini-supervisor is turned off. These results imply that the majority of the performance hit shown in Table 2 was the mini-supervisor. Encryption on the server only doubles the time taken to transmit the job data. This performance hit is unavoidable in any design which introduces encryption to protect data and in most cases will be hidden by the time the printer spends imaging and printing pages.

Table 3 also shows that I underestimated the computational load of Triple-DES encryption. A 386 SX/20 is not just not fast enough to transmit data at the same speed as the other components of the system. A production implementation of the mini-supervisor would need to either have a much faster processor or a DES hardware chip. However, because DES

chips (and even relatively fast processors) are inexpensive, the mini-supervisor can still be relatively inexpensive in comparison to another server.

7. Conclusion

As more confidential data is moved online, the need for secure online services increases. People send everything from credit card numbers to corporate memos over networks, often unaware of whether the data has been securely transmitted or even that it needs to be. Fortunately, many administrators have become aware of this problem and now use authentication and encryption to protect services such as e-mail and web forms.

However, one service which has received little attention in terms of encryption is networked printing. Although some vendors have provided printers which print documents into locked bins or require a code to be entered before they will print, if the document data is transmitted unencrypted over a network, none of these measures will protect the data from being copied in transmission and printed somewhere else.

In response to this problem, departments within the Massachusetts Institute of Technology have asked MIT's Information Systems to design and implement a secure networked printing system for the Athena Computing Environment, so that documents containing confidential data can be printed to printers on MITnet. In order for this system to be used by existing and new hardware, the system cannot depend on any specific printer hardware or vendor to implement encryption of document data.

For my thesis, I designed and built a prototype of such a system, using the concept of mini-supervisors with LPRng, an implementation of the Line Printer Daemon (LPD) protocol. Although the prototype's performance is worse than expected, this is primarily a result of poor hardware choices. With more appropriate hardware and software, a production implementation of a secure printing system which uses mini-supervisors appears to be the ideal solution for the Athena Computing Environment.

References

[1] Adobe Systems Incorporated. *PostScript Language Reference Manual (Second Edition).* Addison Wesley Publishing Company. Reading, MA. 1990.

[2] Academic Technology Services. Net-Print: Welcome to Net-Print. Cornell University. On-line at: http://www.cit.cornell.edu/cit-pubs/net-print/index.html.

[3] D. Bier. *Inside NetWare 4.1.* New Riders Publishing. Indianapolis, IN. 1994.

[4] S. Bradner, R. Cowles, J. Ferrato, S. Hall, T. Head, T. Hanss, R. Knight, C. Lynch, M. S. Lynn, A. Margulies, M. Resmer, L. Sewell, C. M. Taylor, J. Wooden, and S. Worona. "CUPID Protocols and Services (Version 1): An Architectural Overview." Prepared for the Coalition for Networked Information. Cornell University. November 1992.

[5] Digi International. "Digi FastPort Family: Product Profile." Digi International. Online at: http://www.digibd.com/prodprofiles/profiles/fstprtfm/fstprtfm.html.

[6] S. Gurnani. "CUPID Protocol Version 1.0. Revision #3." Cornell University. February 1994. On-line at: http://oitnext.cit.cornell.edu/CUPID/protocol1.0.html.

[7] B. Handspicker, R. Hart and M. Roman. "The Athena Palladium Print System." Open Software Foundation. February 1989.

[8] Hewlett-Packard Corporation. "HP Unveils Industry's Lowest-Priced Multiprotocol Print Server." Hewlett-Packard Corporation. Online at: http://www.hp.com/rnd/whatsnew/releases/pr0506/pr0506.htm. 1996.

[9] M. Hojnowski. Senior Programmer/Analyst at Cornell University. Personal Communication. October 10, 1996.

[10] IBM Corporation. "IBM Introduces First Industrial-Strength Print Management Software for Distributed Environments." September 1995. Online at: http://www.can.ibm.com/ibmprinters/psmrel.html.

[11] J. Isaacson, S. Minai, S. Neiterman, A. Oakland, T. Regan, J. Repa, J. Schiller, S. Thorne, T. Ts'o. Secure Printing Position Paper. Massachusetts Institute of Technology Infrastructure Readiness Team. June 1995. Online at: http://web.mit.edu/reeng/www/rready/printre3.txt.

[12] L. McLaughlin III, Editor. "Line Printer Daemon Protocol." The Wollongong Group Network Printing Group Request for Comments: 1179. August 1990.

[13] Novell Corporation. "Novell Distributed Print Service White Paper". May 1996. Online at: http://www.netware.com/discover/wpndpstc.htm.

[14] SAP AG. "Secure Network Communications -- Integrating R/3 into Network Security Products." 1996. Online at: http://www.sap-ag.de/mall/newstand/litera/pdf/wp_snc_e.pdf.

[15] B. Schneier. *Applied Cryptography.* John Wiley & Sons, Inc. New York, NY. 1994.

[16] Warp Nine Engineering. "IEEE 1284 Parallel Port Information Brought to you by: Warp Nine Engineering" 1994. Online at: http://www.fapo.com/ieee1284.htm.

[17] S. Worona. "CUPID: Breaking the Barriers to Network Publishing." Cornell University. 1995. Online at: http://oitnext.cit.cornell.edu/CUPID/bbnp.html.

[18] Xerox Corporation. "Digital Equipment, SunSoft and Xerox form a Strategic Alliance to Deliver Platform Independent Printing Services and Applications to the Industry." September 1995. Online at: http://www.xerox.com/PR/NR950912-Printxchange.html.

[19] Xerox Corporation. "Meeting the Challenge of Managing Client/Server Document Distribution and Printing: Printxchange White Paper." September 1995. Online at: http://www.xerox.com/PR/NR950912-Whitepaper.html.

[20] Xerox Corporation. "Xerox Platform Provides Industry's First Framework for Integrating Print-On-Demand Services." News from Xerox Corporation Public Relations Office. Hutchins/Y&R, Rochester, N.Y. April 1996.

Appendix A:
Source Code Specific to LPRng

File Name: lp.h

```
/***************************************************************************
 * LPRng - An Extended Print Spooler System
 *
 * Copyright 1988-1997, Patrick Powell, San Diego, CA
 *     papowell@sdsu.edu
 * See LICENSE for conditions of use.
 *
 ***************************************************************************
 * MODULE: lp.h
 * PURPOSE: general type definitions that are used by all LPX facilities.
 * $Id: lp.h,v 3.15 1997/03/24 00:45:58 papowell Exp papowell $
 ***************************************************************************/

#ifndef _LP_H_
#define _LP_H_ 1
#ifndef EXTERN
#define EXTERN extern
#endif

/*****************************************************************
 * get the portability information and configuration
 *****************************************************************/

#include "portable.h"

/*****************************************************************
 * Global variables and routines that will be common to all programs
 *****************************************************************/

/*************************************************
 * Internationalisation of messages, using GNU gettext
 *************************************************/

#if HAVE_LOCALE_H
# include <locale.h>
#endif

#if ENABLE_NLS
# include <libintl.h>
# define _(Text) gettext (Text)
#else
# define _(Text) Text
#endif
#define N_(Text) Text

/*****************************************************************
 * strncpy/strncat(s1,s2,len)
 * The bsdi-compat versions of strncat and strncpy
 * only copy up to the end of string, and a terminating 0.
 *****************************************************************/
```

```
#define safestrncat( s1, s2 ) strncat(s1,s2,sizeof(s1)-strlen(s1)-1)
#define safestrncpy( s1, s2 ) strncpy(s1,s2,sizeof(s1));

/* VARARGS3 */
#ifdef HAVE_STDARGS
int     plp_snprintf (char *str, size_t count, const char *fmt, ...);
int     vplp_snprintf (char *str, size_t count, const char *fmt, va_list arg);
#else
int plp_snprintf ();
int vplp_snprintf ();
#endif


/* VARARGS3 */
#if !defined(HAVE_SETPROCTITLE) || !defined(HAVE_SETPROCTITLE_DEF)
#  ifdef HAVE_STDARGS
void setproctitle( const char *fmt, ... );
#  else
void setproctitle();
#  endif
#endif

#define STR(X) #X

/**************************************************************************
 * Pathname handling -
 **************************************************************************/

struct dpathname{
        char pathname[MAXPATHLEN];
        int pathlen;
};
EXTERN struct dpathname *Tempfile;              /* temporary file */

/**************************************************************************
 * tokens, parameters, and keywords
 * The struct token{} information is used to parse input lines and
 *    extract values.  The start field is the start of the token;
 *    the length field is the length.
 *
 * The struct keywords{} information is used to represent name/value
 *   pairs of information.
 *   The keyword field is a pointer to a string, i.e.- the keyword;
 *   The type field indicates the type of value we have:
 *     INTEGER is integer value,  STRING is string value,
 *     LIST is a list of values,  which is actually an array of pointers
 *     to strings.
 *   The variable field is the address of a variable associated with the
 *     keyword
 *   The maxval field holds the maximum value (length? magnitude?) of variable
 *   The flags field is for holding flags associated with the variable
 **************************************************************************/

enum key_type { FLAG_K, INTEGER_K, STRING_K, LIST_K };

struct token {
        char *start;
        int length;
```

```
};

struct keywords{
    char *keyword;              /* name of keyword */
    enum key_type type; /* type of entry */
    void *variable;             /* address of variable */
        int  maxval;            /* maximum length or value of variable */
                                    /* also used to suppress clearing
value */
        int  flag;                      /* flag for variable */
};
/*************************************************************
 * BUFFER SIZES
 *
 * A common size is 1024 bytes;
 * However,  it appears that this is overkill for most purposes.
 * 180 bytes appears to be satisfactory for a line,
 * 512 for a small buffer, 1024 for a large buffer
 *
 *************************************************************/

#define LINEBUFFER 180
#define SMALLBUFFER 512
#define LARGEBUFFER 1024

/*************************************************************
 * General variables use in the common routines;
 * while in principle we could segragate these, it is not worth it
 * as the number is relatively small
 *************************************************************/
extern char *Copyright[];       /* copyright info */
EXTERN int Init_done;           /* initialization done */
EXTERN char *Logname;           /* Username for logging */
/* EXTERN char *Host;                   / * Hostname */
EXTERN char *ShortHost;         /* Short hostname for logging */
EXTERN char *FQDNHost;          /* FQDN hostname */
EXTERN char *Localhost;         /* Name to be used for localhost lookup */
EXTERN char *Printer;           /* Printer name for logging */
EXTERN char *Orig_printer;      /* Printer name for logging */
EXTERN char *Queue_name;        /* Queue name used for spooling */
EXTERN char *Forwarding;        /* Forwarding to remote host */
EXTERN char *Classes;           /* Classes for printing */
EXTERN int Destination_port;    /* Destination port for connection */
EXTERN char *Server_order;      /* order servers should be used in */
EXTERN int Max_servers;         /* max servers currently active */
EXTERN int Max_servers_active;  /* maximum number of servers active */
EXTERN int IPV6Protocol;        /* IPV4 or IPV6 protocol */
extern int AF_Protocol;         /* AF protocol */
EXTERN char* Kerberos_service;  /* kerberos service */
EXTERN char* Kerberos_keytab;   /* kerberos keytab file */
EXTERN char* Kerberos_life;     /* kerberos lifetime */
EXTERN char* Kerberos_renew;    /* kerberos newal time */
EXTERN char* Kerberos_server_principle; /* kerberos server principle */
EXTERN int Poll_time;          /* time in secs between starting up all servers */

/*************************************************************
 * Command line options and Debugging information
 * Getopt is a modified version of the standard getopt(3) command
 *  line parsing routine. See getopt.c for details
```

```
 ****************************************************************/

/* use this before any error printing, sets up program Name */
int Getopt( int argc, char *argv[], char *optstring );
extern int Optind, Opterr;
extern char *Optarg;
extern char *Name;                          /* program name */

/*****************************************************************************
 * Dynamic memory allocation control
 * the idea is that you allocate lists or blocks,  and then you keep track
 * of how much you have allocated
 *****************************************************************************/

struct malloc_list{
        char **list;    /* array of pointers to allocated blocks */
        int count;              /* number of entries in list */
        int max;                /* max number of entries in list */
        int size;               /* size of each entry in list: error checking */
};

/* list of control files for status information */
EXTERN struct malloc_list C_files_list;

/*****************************************************************************
 *      Control file format
 *      First character is kind of entry, remainder of line is
 *          the argument.
 *
 *              1 -- "R font file" for troff -ignore
 *              2 -- "I font file" for troff -ignore
 *              3 -- "B font file" for troff -ignore
 *              4 -- "S font file" for troff -ignore
 *              C -- "class name" on banner page (priority)
 *              D -- "Date" job submitted
 *              H -- "Host name" of machine where lpr was done
 *              I -- "indent" amount to indent output
 *              J -- "job name" on banner page
 *              L -- "literal" user's name to print on banner
 *              M -- "mail" to user when done printing
 *              N -- "name" of file (used by lpq)
 *              P -- "Person" user's login name
 *              Q -- "Queue Name" used for spooling
 *              R -- account id  for charging
 *              U -- "unlink" name of file to remove after we print it
 *              W -- "width" page width for PR
 *              Z -- xtra options to filters
 *
 *              Lower case letters are formats
 *              f -- "file name" name of text file to print
 *              l -- "file name" text file with control chars
 *              p -- "file name" text file to print with pr(1)
 *              t -- "file name" troff(1) file to print
 *              n -- "file name" ditroff(1) file to print
 *              d -- "file name" dvi file to print
 *              g -- "file name" plot(1G) file to print
 *              v -- "file name" plain raster file to print
 *              c -- "file name" cifplot file to print
 *
```

```
*/

/***************************************************************
 * Control File Information
 *
 *  Patrick Powell Sat Apr  8 08:04:28 PDT 1995
 *
 * The struct data_file{} is used to record the data files in
 * a print job.
 *  - name field is the 'N' option in the control file
 *  - datafile field is the name of the data file
 *  - size field is the size (in bytes) of the data file
 *
 * The struct control_file{} is used to record control file information
 * about a print job.   This data structure is meant to be used
 * in most of the LPR software,  and has a slight amount of overkill
 * in terms of its facilities.
 * 1. the name and stat fields are used when determining job order
 *    and priority.
 * 2. the options and option_count fields are used when parsing the
 *    control file, and determining the various options in the file.
 * 3. the capoptions[] array points to the various options starting
 *    with capital letters and the digitoptions with digit letters
 * 4. the datafile[] field points to an array of data file structures,
 *    each of which corresponds to a data file in a job.
 * 5. the buffer field points to a block of memory which holds the
 *    control file.
 *
 * Commentary:
 *  It is a lot easier to simply read a control file into memory,
 *  then parse it and put pointers to the various things that are
 *  needed.  Note that when you do a scan of a directory for job
 *  information,  you will need to read the control file anyways.
 *  Thus,  if you simply read them into memory,  you will have
 *  all of the information available,  and only need to read them
 *  once until they change.
 *
 *  Rescanning the spool directory becomes almost trivial;  you simply
 *  stat each file in order,  check to see if it is in the list,
 *  and check the stat information.  If the entry is not in the list
 *  then you add it.  If an entry has changed or is not found on the
 *  scan,  then you remove the current entry and add it again.
 *
 *  When transferring a job (forwarding) to another LPD server,  you
 *  may have to reformat the information in the control file. Note
 *  that this usually consists of reordering the fields and/or
 *  eliminating some fields.  By having a cracked version of this
 *  file available you make life simpler.
 *
 *  Note that when you send the control file,  you will have to
 *  calculate the lengths of all of the fields to send.  You can
 *  do this by rescanning the options[] array and using strlen()
 *  on each of the lines in the control file.  Note that this
 *  has to be done only once,  so it is not an insurmountable overhead.
 *
 *  Critics of this method point out that you will now need to do a
 *  'line by line' write to the socket rather than doing a block write;
 *  many of these forget that you are dealing with TCP/IP, which will
 *  invoke the Van Jacobsen and Slow Start algorithm,  and transfer
```

```
 *  maximum size blocks (see RFC1720, RFC1722, and the references
 *  listed in them).  So there is effectively no impact on the TCP/IP
 *  throughput.  If you are really worried about overhead,  you can
 *  play games with dup(), fdopen(),  and fprintf(),  and use the
 *  STDIO library buffering functions.   I have experimented with
 *  this and found that there was only 3% performance change in the
 *  PLP performance;  apparently the cost of calls to dirent() overwhelm the
 *  calls to 'write',  so you really don't gain much for the huge effort
 *  in coding.
 *
 *  One of the interesting effects of this strategy is to cause a change
 *  in the way that status information for queues is obtained.  In
 *  most LPD implemtations,  the LPD server will fork a process that
 *  will then scan the spool queues.  Each request has a rather
 *  substantial overhead,  and has led to some rather nasty denial of
 *  service network attacks.  A different method would be to allow
 *  the LPD daemon to keep the information,  and then when a
 *  request arrives to rescan the spool queues.  Once the spool queues
 *  are rescanned,  then the LPD daemon would fork a process to handle
 *  the reporting of the information.   This would make the LPD
 *  daemon process large (in terms of memory),  but reduce the load
 *  on systems which are suffering from denial of service attacks.
 *  An alternative to this method would be to put a throttle on the
 *  number of processes allowed to be created for reporting spool queue
 *  information.
 ****************************************************************/

struct data_file {
        char openname[MAXPATHLEN];              /* full pathname of datafile */
        char transfername[LINEBUFFER];  /* transfername of data file - short */
        char original[LINEBUFFER];      /* original name of data file - short */
        char Ninfo[LINEBUFFER]; /* 'N' Option information: userfile, (stdin) */
        char Uinfo[LINEBUFFER]; /* 'U' Option information */
        int format;                     /* format */
        int fd;                         /* file descriptor for reading */
        struct stat statb;      /* stat information */
        int flags;                      /* flags */
        int found;                      /* job found in control file and sent */
#define PIPE_FLAG               0x01    /* pipe */
        int copies;                     /* if non-zero, this is the transfered
copy */
       off_t offset;           /* offset from the start of the block format file
*/
        off_t length;           /* length file */
};


struct hold_file {
        int  attempt;                   /* number of print attempts */
        int  not_printable;             /* printable status */
        int  held_class;                /* held class for printing */
        long priority_time;             /* priority time */
        long hold_time;                 /* hold time */
        long remove_time;               /* removal time */
        long done_time;                 /* finished time */
        long routed_time;               /* routed to destinations */
        int  active;                    /* pid of server if it is active */
        int  receiver;          /* pid of receiver if it is active, -1 if all
arrived */
```

```
        char redirect[LINEBUFFER];      /* redirection */
};

struct control_file {
        char openname[MAXPATHLEN];      /* full pathname of control file - used
in open/close */
        char transfername[LINEBUFFER];  /* control file - used in transfers  */
       char original[LINEBUFFER];     /* original file name - received from LPR
or LPD */
        int  number;                    /* job number */
        int  number_len;                /* number of digits in job number */
        int  max_number;                /* maximum job number */
        int  recvd_number;              /* original number of job */
        int  recvd_number_len;  /* original digits of number of job */
        int  jobsize;                   /* size of job in bytes */
        int  copynumber;                /* copy number */
       char filehostname[LINEBUFFER];  /* hostname part of the control file name
*/
        struct stat statb;              /* stat of control file information */
        char hold_file[MAXPATHLEN];     /* full pathname of hold file */
        struct stat hstatb;             /* stat of hold file information */
        int priority;                   /* priority from the job ID */
       int flags;                            /* flags used to indicate an error
or status */
#define LAST_DATA_FILE_FIRST 0x2
       int found;                           /* found in the directory structure
*/
        int remove_on_exit;             /* if remove job files on exit */
        char auth_id[LINEBUFFER];       /* authenticated information */
        char forward_id[LINEBUFFER];
        char authtype[LINEBUFFER];      /* authentication type */
#define BAD_FLAG       0x01
#define OLD_FLAG       0x02
#define ACTIVE_FLAG    0x04
#define FOUND_FLAG     0x08
        /* block format for files */
        int block_format;               /* block format flag */
        off_t start;                    /* offset from start */
        off_t len;                              /* length of the control file */
        char identifier[LINEBUFFER];/* identifier */
        char error[LINEBUFFER]; /* error message for bad control file */
       int  control_info;             /* number of lines before data file infor-
mation
                                                              in control file */
        char *capoptions[26];   /* capital letter options */
        char *digitoptions[10]; /* digit options */
        char *cf_info;                  /* control file image */
        /* from the hold file */
       int   destination_info_start;   /* destination information starts here */
        struct hold_file hold_info;
        struct malloc_list control_file_image;  /* control file name and image
(buffers) */
      struct malloc_list data_file_list;            /* allocated area for data
file list */
      struct malloc_list control_file_lines;        /* allocated area for line
list */
        struct malloc_list control_file_copy;   /* allocated area for control
file copy */
       struct malloc_list control_file_print;  /* printable copy of control file
```

```
*/
       struct malloc_list hold_file_info;      /* allocated area for hold file */
        struct malloc_list hold_file_lines;     /* allocated area for hold file
lines */
        struct malloc_list hold_file_print;             /* printable copy of con-
trol file */
       struct malloc_list destination_list;    /* allocated area for destination
list */
};

struct destination {
        char destination[LINEBUFFER];   /* destination of job */
        char identifier[LINEBUFFER];    /* new identifier of job */
        char error[LINEBUFFER];                 /* error message for bad control
file */
        int arg_start;                                  /* option lines to add/
change */
        int arg_count;                                  /* number of lines */
       int copies;                                          /* number of copies
*/
        int copy_done;                                  /* copies done */
       int status;                                          /* status of copy */
        int hold;                                           /* held */
        int active;                                         /* active */
        int done;                                           /* active */
        int attempt;                                    /* attempt */
        int ignore;                                         /* ignore */
       int sequence_number;                    /* sequence number to add to job */
        int not_printable;
};

/************************************************************************
 * ACKs from the remote host on job transfers
 ************************************************************************/
#define ACK_SUCCESS 0   /* succeeded; delete remote copy */
#define ACK_STOP_Q  1   /* failed; no spooling to the remote queue */
#define ACK_RETRY   2   /* failed; retry later */
#define ACK_FAIL    3   /* failed; bad job */

/*
 * macros for the names and maximum lengths for the various fields
 */

#define M_DEFAULT               131                             /* default limit
on line length */
#define IDENTIFIER capoptions['A'-'A']  /* LPRng - unique job ID */
#define M_IDENTIFIER    131                             /* default limit on line
length */
#define CLASSNAME capoptions['C'-'A']   /* RFC: 31 char limit */
#define M_DATE                  31
#define DATE      capoptions['D'-'A']   /* LPRng - date job started */
#define M_CLASSNAME             31
#define FROMHOST  capoptions['H'-'A']   /* RFC: 31 char limit */
#define M_FROMHOST              31
#define INDENT    capoptions['I'-'A']   /* RFC: number */
#define M_INDENT                31
#define JOBNAME   capoptions['J'-'A']   /* RFC: 99 char limit */
#define M_JOBNAME               99
#define BNRNAME   capoptions['L'-'A']   /* RFC: ?? char limit */
```

```
#define M_BNRNAME               31
#define FILENAME  capoptions['N'-'A']   /* RFC: 131 char limit */
#define M_FILENAME              131
#define MAILNAME  capoptions['M'-'A']   /* RFC: ?? char limit */
#define M_MAILNAME              131
#define M_NAME                  131                             /* RFC: 131 char
limit on 'N' entries */
#define LOGNAME   capoptions['P'-'A']   /* RFC: 31 char limit */
#define M_LOGNAME               31
#define QUEUENAME capoptions['Q'-'A']   /* PLP: 31 char limit */
#define M_QUEUENAME             31
#define ACCNTNAME capoptions['R'-'A']   /* PLP: info for accounting: 131 */
#define M_ACCNTNAME             131
#define SLINKDATA capoptions['S'-'A']   /* RFC: number " " number */
#define M_SLINKDATA             131
#define PRTITLE   capoptions['T'-'A']   /* RFC: 79 char limit */
#define M_PRTITLE               79
#define UNLNKFILE capoptions['U'-'A']   /* RFC: flag */
#define M_UNLNKFILE             131
#define PWIDTH    capoptions['W'-'A']   /* RFC: number */
#define M_PWIDTH                31
#define ZOPTS     capoptions['Z'-'A']   /* PLP */
#define M_ZOPTS                 131
#define FONT1     digitoptions['1'-'0'] /* RFC: ?? char limit */
#define FONT2     digitoptions['2'-'0'] /* RFC: ?? char limit */
#define FONT3     digitoptions['3'-'0'] /* RFC: ?? char limit */
#define FONT4     digitoptions['4'-'0'] /* RFC: ?? char limit */
#define M_FONT          131

/*****************************************************************
 * Command file option checking:
 *  We specify the allowed order of options in the control file.
 *  Note that '*' is a wild card
 *   Berkeley-            HPJCLIMWT1234
 *   PLP-                 HPJCLIMWT1234*
 *****************************************************************/
/* error codes for return values */

#define LINK_OPEN_FAIL          -1              /* open or connect */
#define LINK_TRANSFER_FAIL      -2              /* transfer failed */
#define LINK_ACK_FAIL           -3              /* non-zero ACK on operation */
#define LINK_FILE_READ_FAIL     -4              /* read from a file failed */
#define LINK_LONG_LINE_FAIL     -5             /* a line was too long to read */
#define LINK_BIND_FAIL          -6              /* cannot bind to port */
#define LINK_PERM_FAIL         -7             /* permission failure, remove job
*/

/*****************************************************************
 * LPD Protocol Information
 *****************************************************************/
/*****************************************************************
 * Request types
 * A request sent to the LPD daemon has the format:
 * \Xprinter [options],  where \X is a single character or byte value.
 * The following are the values and commands
 *****************************************************************/

#define REQ_START   1   /* start printer */
#define REQ_RECV    2   /* transfer a printer job */
```

```
#define REQ_DSHORT  3   /* print short form of queue status */
#define REQ_DLONG   4   /* print long form of queue status */
#define REQ_REMOVE  5   /* remove jobs */
#define REQ_CONTROL 6   /* do control operation */
#define REQ_BLOCK   7   /* transfer a block format print job */
#define REQ_SECURE  8   /* secure command transfer */
#define REQ_VERBOSE 9   /* verbose status information */

#define AUTHENTICATE 1       /* \1authentication information\n */
#define CONTROL_FILE 2       /* \2<count> <cfname>\n */
#define DATA_FILE    3       /* \3<count> <dfname>\n */

/*****************************************************************
 * The expanded printcap entry is created by running through
 * all of the printcap entries and extracting the required
 * information from them.  The raw entries are extracted from
 * the printcap files themselves;  the expanded entries are
 * the  combined information.
 *  Note that the raw_list is the address of the pointer to the
 *  start of the raw list information;  nameindex and optionindex
 *  are the indices into this raw list.  When the printcap
 *  is expanded,  the namelines and line entries will be used
 *  to hold the array of pointers to names and entries.  This is
 *  done because the original raw list can grow and may need to
 *  be relocated in memory.  Once it has been relocated, the
 *  raw_list value is set to 0.
 *****************************************************************/

struct printcap_entry {
        char **names;           /* start of array of printcap names */
        int namecount;          /* number of names  */
        char **options;         /* start of array of options */
        int optioncount;        /* number of options */
        char *key;                      /* key for checking spool dirs */
        int status_done;        /* status reported */
        int checked;            /* printcap already checked */
        struct malloc_list namelines;   /* names and options in list */
        struct malloc_list lines;       /* names and options in list */
                /* count is number of lines in entry */
                /* list is an array of char */
};

/*
 * each printcap file is read into a buffer, and then
 * parsed for lines and printcap entries.  The include
 * processing is done at this point;  the file and
 * include file data is stored in the files data structure
 *
 * The file data is scanned,  and broken down into lines and
 * then into entries.  The entries field will be an array of
 * pointers to the first character in each field.
 *
 * After the data file has been scanned for entries, it
 * will be rescanned for printcap entries.  Each one of
 * these will be put into the printcaps field.
 */

struct file_entry {
        int initialized;                        /* initialized flag */
```

```
        struct malloc_list files;       /* storage for file data and include */
                /* list is an array of char * (must be freed) */
        struct malloc_list entries;     /* storage for entries */
                /* list is an array of char ** */
        struct malloc_list filters;     /* storage for filters */
                /* list is an array of char * (must be freed ) */
        struct malloc_list printcaps;   /* storage for printcaps */
                /* list is an array of struct printcap */
       struct malloc_list expanded_str;       /* storage for expanded strings */
};

/*************************************************************
 * LPD Control file information
 * control file has the format:
 *   spooling on/off
 *   printing on/off
 *************************************************************/

EXTERN int Printing_disabled;   /* No printing */
EXTERN int Spooling_disabled;   /* No spooling */

/*************************************************************
 * Process forking support
 * Idea: we generate the command line for the process from pieces;
 *   we check to make sure that each piece has no problems.
 * The struct args {} data structure is used to set this up.
 *************************************************************/

struct filter {
        int pid;                /* PID of filter process */
        int input;              /* input file descriptor */
        int output;             /* output file descriptor of filter */
        char *cmd;              /* command string - malloc */
        char *copy;             /* copy of command string - malloc */
        struct dpathname exec_path;     /* pathname */
        struct malloc_list args;                /* argument list */
        struct malloc_list env;                 /* environment variables */
        struct malloc_list envp;                /* pointers */
};


/*************************************************************
 * Error Printing
 * errormsg.c file
 *************************************************************/

EXTERN int Errorcode;           /* Exit code for an error */
EXTERN int Interactive;         /* Interactive or Server mode */
EXTERN int Verbose;                     /* Verbose logging mode */
EXTERN int Echo_on_fd;          /* Echo Error on fd */
EXTERN int Syslog_fd;           /* syslog device if no syslog() facility */
EXTERN int Status_fd;           /* status file file descriptor */

/*************************************************************
 * Routing Support
 *************************************************************/

EXTERN struct destination *Destination; /* Destination for routing */
```

```
#define malloc_or_die(x,y)  if (((x) = malloc((unsigned) y)) == 0) { \
        Malloc_failed((unsigned)y); \
}

#define roundup_2(x,y) ((x+((1<<y)-1))&~((1<<y)-1))

/**************************************************************************
 * Subserver information
 *  used by LPQ and LPD to start subservers
 **************************************************************************/

struct server_info{
    char *name;         /* printer name for server */
    pid_t pid;          /* pid of server processes */
    int status;         /* last exit status of the server process */
    int initial;        /* 1 = initial c leanup server */
        int printing_disabled;  /* printing disabled */
    unsigned long time; /* time it terminated */
        char transfername[LINEBUFFER];  /* transfername of file */
        struct dpathname spooldir;      /* spool directory */
};

/*****************************************************************
 * Include files that are so common they should be included anyways
 *****************************************************************/

#include "bsd-compat.h"
#include "errormsg.h"
#include "debug.h"
#include "utilities.h"

/**************************************************************************
 * Variables whose values are defined by entries in the printcap file
 *
 * We extract these from the printcap file when we need them
 **************************************************************************/

/*
 * note: most of the time the Control directory pathname
 * and the spool directory pathname are the same
 * However, for tighter security in NFS mounted systems, you can
 * make them different
 */

EXTERN int Is_server;                           /* LPD sets to non-zero */
EXTERN int Auth_from;                           /* LPD sets to type of authenti-
cation */
EXTERN struct dpathname *CDpathname;    /* control directory pathname */
EXTERN struct dpathname *SDpathname;    /* spool directory pathname */

EXTERN int Accounting_check; /* check accounting at start */
EXTERN char* Accounting_end;/* accounting at start (see also af, la, ar) */
EXTERN char* Accounting_file; /* name of accounting file (see also la, ar) */
EXTERN int Accounting_remote; /* write remote transfer accounting (if af is set)
*/
EXTERN char* Accounting_start;/* accounting at start (see also af, la, ar) */
EXTERN char* Allow_class; /* allow these classes to be printed */
EXTERN int Always_banner; /* always print banner, ignore lpr -h option */
EXTERN int Auto_hold;    /* automatically hold all jobs */
```

```
EXTERN int Backwards_compatible; /* backwards-compatible: job file format */
EXTERN int Backwards_compatible_filter; /* backwards-compatible: filter parame-
ters */
EXTERN char* Banner_end;         /* end banner printing program overrides bp */
EXTERN int Banner_last; /* print banner after job instead of before */
EXTERN char* Banner_line;        /* short banner line sent to banner printer */
EXTERN char* Banner_printer; /* banner printing program (see ep) */
EXTERN char* Banner_start;       /* start banner printing program overrides bp */
EXTERN int Baud_rate; /* if lp is a tty, set the baud rate (see ty) */
EXTERN char* Bounce_queue_dest; /* destination for bounce queue files */
EXTERN int Lpr_bounce; /* allow LPR to do bounce queue filtering */
EXTERN int Clear_flag_bits; /* if lp is a tty, clear flag bits (see ty) */
EXTERN int Clear_local_bits; /* if lp is a tty, clear local mode bits (see ty) */
EXTERN char* Comment_tag; /* comment identifying printer (LPQ) */
EXTERN int Connect_grace; /* grace period for reconnections */
EXTERN char* Control_dir; /* Control directory */
EXTERN char* Control_filter; /* Control filter */
EXTERN int Cost_factor; /* cost in dollars per thousand pages */
EXTERN char* Default_auth;       /* default authentication type */
EXTERN char* Default_logger_port;       /* default logger port */
EXTERN char* Default_logger_protocol;   /* default logger protocol */
EXTERN char* Default_priority;  /* default priority */
EXTERN char* Default_format;    /* default format */
EXTERN int Direct_read; /* filter reads directly from a file */
EXTERN int FF_on_close; /* print a form feed when device is closed */
EXTERN int FF_on_open; /* print a form feed when device is opened */
EXTERN char* Filter_control; /* filter control file */
EXTERN char* Force_queuename; /* force the use of this queue name */
EXTERN char* Form_feed; /* string to send for a form feed */
EXTERN char* Formats_allowed; /* valid output filter formats */
EXTERN int Forwarding_off; /* if true, no forwarded jobs accepted */
EXTERN int Generate_banner; /* generate a banner when sending to remote */
EXTERN int Hold_all;     /* hold all jobs */
EXTERN char* IF_Filter; /* filter command, run on a per-file basis */
EXTERN int Is_Mini_Supervisor; /* true if the lp variable refers to a mini-super-
visor*/
EXTERN char* Leader_on_open; /* leader string printed on printer open */
EXTERN int Local_accounting; /* write local printer accounting (if af is set) */
EXTERN char* Local_permission_file; /* additional permissions file for this
queue */
EXTERN int Lock_it; /* lock the IO device */
EXTERN char* Log_file; /* error log file (servers, filters and prefilters) */
EXTERN char* Logger_destination; /* logger process host */
EXTERN int Long_number; /* long job number (6 digits) */
EXTERN char* Lp_device; /* device name or lp-pipe command to send output to */
EXTERN int Max_copies; /* maximum copies allowed */
EXTERN int Max_job_size; /* maximum job size (1Kb blocks, 0 = unlimited) */
EXTERN int Mini_Supervisor_Encrypt; /* true if mini-supervisor should encrypt */
EXTERN int Min_printable_count; /* minimum printable characters for printable
check */
EXTERN char* Minfree; /* minimum space (Kb) to be left in spool filesystem */
EXTERN int NFS_spool_dir; /* spool dir is on an NFS file system (see rm, rp) */
EXTERN char* New_debug; /* debug level set for queue handler */
EXTERN int No_FF_separator; /* suppress form feeds separating multiple jobs */
EXTERN int Nonblocking_open; /* nonblocking open on io device */
EXTERN char* OF_Filter; /* output filter, run once for all output */
EXTERN int Page_length; /* page length (in lines) */
EXTERN int Page_width; /* page width (in characters) */
EXTERN int Page_x; /* page width in pixels (horizontal) */
```

```
EXTERN int Page_y; /* page length in pixels (vertical) */
EXTERN char* Pr_program; /* pr program for p format */
EXTERN int Read_write; /* open the printer for reading and writing */
EXTERN char* RemoteHost; /* remote-queue machine (hostname) (with rm) */
EXTERN char* RemotePrinter; /* remote-queue printer name (with rp) */
EXTERN char* Routing_filter; /* filter to determine routing of jobs */
EXTERN int Save_on_error; /* save this job when an error */
EXTERN int Save_when_done; /* save this job when done */
EXTERN char* Server_authentication_command; /* authentication command */
EXTERN char* Server_names; /* names of servers for queue (with ss) */
EXTERN char* Server_queue_name; /* name of queue that server serves (with sv) */
EXTERN int Send_block_format; /* send block of data */
EXTERN int Send_data_first; /* send data files first */
EXTERN int Set_flag_bits; /* like `fc' but set bits (see ty) */
EXTERN int Set_local_bits; /* like `xc' but set bits (see ty) */
EXTERN int Short_banner; /* short banner (one line only) */
EXTERN char* Spool_dir; /* spool directory (only ONE printer per directory!) */
EXTERN int Spread_jobs; /* spread job numbers out by this factor */
EXTERN char* Status_file; /* printer status file name */
EXTERN char* Stty_command; /* stty commands to set output line characteristics */
EXTERN int Suppress_copies; /* suppress multiple copies */
EXTERN int Suppress_header; /* suppress headers and/or banner page */
EXTERN char* Trailer_on_close; /* trailer string to print when queue empties */
EXTERN int Use_date;            /* put date in control file */
EXTERN int Use_queuename;       /* put queuename in control file */
EXTERN int Use_identifier;      /* put identifier in control file */
EXTERN int Use_shorthost;      /* Use short hostname in control file information
*/
EXTERN char* Use_auth;          /* clients use authentication */
EXTERN int Use_auth_flag;       /* clients forcing authentication */
EXTERN char* Forward_auth;      /* server use authentication when forwarding */
EXTERN char* Xlate_format;      /* translate format ids */
EXTERN char* Pass_env;          /* pass these environment variables */

/***************************************************************************
 * Configuration information
 ***************************************************************************/
EXTERN char * Architecture;
EXTERN int Allow_getenv;
EXTERN char * BK_filter_options;        /* backwards compatible filter options */
EXTERN char * BK_of_filter_options;     /* backwards compatible OF filter options
*/
EXTERN int Check_for_nonprintable;      /* lpr check for nonprintable file */
EXTERN char * Client_config_file;
EXTERN char *Connect_failure_action;
EXTERN int Connect_interval;
EXTERN int Connect_try;
EXTERN int Connect_timeout;
EXTERN char * Default_banner_printer;   /* default banner printer */
EXTERN char * Default_permission;       /* default permission */
EXTERN char * Default_printer;  /* default printer */
EXTERN char * Default_remote_host;
EXTERN char * Default_tmp_dir;  /* default temporary file directory */
EXTERN char * Server_tmp_dir;   /* default temporary file directory */
EXTERN char * Filter_ld_path;
EXTERN char * Filter_options;
EXTERN char * Filter_path;
EXTERN char * Lockfile;
EXTERN char * Logfile;
```

```
EXTERN char * Lpd_port;
EXTERN char * Lpd_printcap_path;
EXTERN char * Mail_operator_on_error;
EXTERN int Max_status_size;
EXTERN int Min_status_size;
EXTERN char * Minfree; /**/
EXTERN char * Originate_port;
EXTERN char * OF_filter_options;
EXTERN char * Printcap_path;
EXTERN char * Printer_perms_path;
EXTERN char * Sendmail;
EXTERN char *Send_failure_action;
EXTERN int Send_try;
EXTERN int Send_timeout;
EXTERN char * Server_config_file;
EXTERN int Spool_dir_perms;
EXTERN int Spool_file_perms;
EXTERN char *Syslog_device;     /* default syslog device if no syslog() facility
*/
EXTERN int Use_info_cache;
EXTERN char * User_authentication_command;
EXTERN char * Daemon_group;
EXTERN char * Server_user;
EXTERN char * Remote_user;
EXTERN char * Daemon_user;

/***************************************************************************
 * lpr variables
 ***************************************************************************/

EXTERN char *Accntname; /* Accounting name: PLP 'R' control file option */
EXTERN int Binary;      /* Binary format: 'l' Format */
EXTERN char *Bnrname;   /* Banner name: RFC 'L' option */
EXTERN char *Classname; /* Class name:  RFC 'C' option */
EXTERN int Copies;      /* Copies */
EXTERN char *Format;    /* format for printing: lower case letter */
EXTERN char *Font1;     /* Font information 1 */
EXTERN char *Font2;     /* Font information 2 */
EXTERN char *Font3;     /* Font information 3 */
EXTERN char *Font4;     /* Font information 4 */
EXTERN int Indent;      /* indent:      RFC 'I' option */
EXTERN char *Jobname;   /* Job name:    RFC 'J' option */
EXTERN char *Mailname;  /* Mail name:   RFC 'M' option */
EXTERN int No_header;   /* No header flag: no L option in control file */
EXTERN char *Option_order;      /* Option order in control file */
EXTERN char *Printer;   /* printer name */
EXTERN int Priority;    /* Priority */
EXTERN char *Prtitle;   /* Pr title:    RFC 'T' option */
EXTERN int Pwidth;          /* Width paper: RFC 'W' option */
EXTERN int Removefiles;     /* Remove files */
EXTERN char *Username;  /* Specified with the -U option */
EXTERN int Use_queuename_flag;  /* Specified with the -Q option */
EXTERN int Secure;              /* Secure filter option */
EXTERN int Setup_mailaddress;   /* Set up mail address */
EXTERN char *Zopts;     /* Z options */

EXTERN int Filecount;   /* number of files to print */
EXTERN char **Files;    /* pointer to array of file names */
EXTERN int DevNullFD;   /* DevNull File descriptor */
```

```
/************************************************************************
 * Information from host environment and defaults
 ************************************************************************/

EXTERN char *FQDNRemote;    /* FQDN of Remote host */
EXTERN char *ShortRemote;   /* Short form of Remote host */
EXTERN int Foreground;      /* Run lpd in foreground */
EXTERN int Clean;           /* clean out the queues */
EXTERN int Server_pid;      /* PID of server */
EXTERN int Lpd_pipe[2];     /* connection between jobs */
EXTERN int All_printers;    /* show all printers */
EXTERN int Clear_scr;       /* clear screen */
EXTERN int Interval;        /* display interval */
EXTERN int Longformat;      /* Long format */
EXTERN int Displayformat;   /* Display format */
EXTERN int All_printers;    /* show all printers */
EXTERN int Max_status_lines; /* number of status lines */
EXTERN struct keywords Lpd_parms[]; /* lpd parameters */
extern char LPC_optstr[];   /* LPD options */
extern char LPD_optstr[];   /* LPD options */
extern char LPQ_optstr[];   /* LPQ options */
extern char LPR_optstr[];   /* LPQ options */
extern char LPRM_optstr[];  /* LPQ options */
extern struct keywords Lpr_parms[]; /* parameters for LPR */
EXTERN int LP_mode;                 /* LP mode */
EXTERN int Lp_getlist;          /* show default printer */
EXTERN int Lp_sched;            /* print scheduler information */
EXTERN int Lp_status;           /* print scheduler information */
EXTERN int Lp_showjobs;         /* print scheduler information */
EXTERN int Lp_accepting;        /* print scheduler information */
EXTERN int Lp_default;          /* print scheduler information */
EXTERN int Lp_summary;          /* print scheduler information */
EXTERN struct malloc_list Lp_pr_list;

/************************************************************************
 * For printcap and configuration information
 ************************************************************************/
EXTERN struct control_file *Cfp_static; /* control file for job */
EXTERN struct malloc_list Data_files; /* list of data files received */

EXTERN char *Control_debug;


/*****************************************************************
 * Host name and address information
 *  The gethostbyname function returns a pointer to a structure
 *  which contains all the host information.  But this
 *  value disappears or is modified on the next call.  We save
 *  the information  using this structure.  Also,  we look
 *  forward to the IPV6 structure,  where we need a structure
 *  for an address.
 *****************************************************************/

struct host_information{
        struct malloc_list host_names;  /* official name of host is first */
        int host_addrtype;              /* address type */
        int host_addrlength;    /* address length */
        struct malloc_list host_addr_list;      /* address list */
```

```
        char shorthost[LINEBUFFER];
        char *fqdn;
};

EXTERN struct host_information LocalhostIP;     /* IP from localhost lookup */
EXTERN struct host_information RemoteHostIP;    /* IP from localhost lookup */
EXTERN struct host_information HostIP;  /* current host ip */
EXTERN struct host_information LookupHostIP;    /* for searches */
EXTERN struct host_information PermcheckHostIP; /* for searches */

/************************************************************************
 * inet_ntop_sockaddr()
 *  - interface to inet_ntop that does checking
 ************************************************************************/
char *inet_ntop_sockaddr( struct sockaddr *addr, char *str, int len );

/************************************************************************
 * you need a place for this stuff
 ************************************************************************/

void Get_subserver_info( struct malloc_list *servers, char *s);
int Receive_job( int *socket, char *input, int maxlen );
int Receive_secure( int *socket, char *input, int maxlen );
int Receive_block_job( int *socket, char *input, int maxlen );
int Job_status( int *socket, char *input, int maxlen );
int Job_remove( int *socket, char *input, int maxlen );
int Job_control( int *socket, char *input, int maxlen );
void Get_parms(int argc,char *argv[]);
off_t Copy_stdin( struct control_file *cf );    /* copy stdin to a file */
off_t Check_files( struct control_file *cf,  char **files, int filecount );
int Make_job( struct control_file *cf );
void Process_jobs( int *socket, char *input, int maxlen );
void Start_all( void );
int Find_non_colliding_job_number( struct control_file *cfp );
int Scan_block_file( int fd, struct control_file *cfp );
int Check_for_missing_files( struct control_file *cfp,
        struct malloc_list *data_files_list, int temp_fd, char *cf_name,
        char *orig_name, char *authentication );
int Do_perm_check( struct control_file *cfp );
void Do_queue_jobs( char *name );
void Sendmail_to_user( int status, struct control_file *cfp,
    struct printcap_entry *pc );
void Start_new_server( void );

#endif
```

File Name: Makefile.in

```
##########################################################################
# LPRng - An Extended Print Spooler System
#
# Copyright 1988-1997 Patrick Powell, San Diego, California
#     papowell@sdsu.edu
# See LICENSE for conditions of use.
#
##########################################################################
# MODULE: Makefile
# $Id: Makefile.in,v 3.19 1997/03/24 00:45:58 papowell Exp papowell $
##########################################################################
#
# WARNING: This is a makefile for GNU Make or when modified by BSD Make
# the appropriate install script.
# GNU Make uses VPATH and replacement facilities
# BSD Make uses .PATH and replacement facilities

#************** GENERAL CONFIGURATION INFORMATION ************************
# where user commands are installed: lpr, lpq, lprm, lpc.
# Also the SysV emulation commands: lp, lpstat.
# If these paths are /usr/lib, /usr/ucb etc., make sure you
# have backup copies of the system version!
#
#
# These are automatically sedded in by the "configure" script, but you
# may need to change them if you're on a strange version of UNIX.

# any libs that need to be used to get these programs to compile:
#
LIBS=@LIBS@
# LIBS=-lposix -s       # (on A/UX, if the above doesn't work)


# what command to use to "ranlib" or "lorder" the library.
#
RANLIB=@RANLIB@

# the compiler optimisation/debugging flags you wish to use.
#
# what C compiler to use.
#
CC=@CC@

# what C preprocessor to use.
#
CPP=@CPP@ $(CCOPTFLAGS)

# how to install stuff.
#
INSTALLCMD=@INSTALL@

# a shell interpreter that's as close to the POSIX shell as possible.
#
SHELL_PATH=@SHELL@
SHELL = @SHELL@
```

```
SRC=@srcdir@
@SET_MAKE@
LDCC=@LDCC@

# where executables are installed
prefix=@prefix@
exec_prefix=@exec_prefix@
localedir = $(prefix)/@DATADIRNAME@/locale
INSTALL_BIN =     @exec_prefix@/bin
# where daemons are installed: lpd
#INSTALL_LIB =     @prefix@/lib
INSTALL_LIB =     @prefix@/bin
# where maintenance commands are installed: checkpc, setstatus
INSTALL_MAINT =   @exec_prefix@/bin

CF := @CFLAGS@ @DEFS@ -DLOCALEDIR=\"$(localedir)\"

#********************** End of configure modifiable settings ************

#********************** OPTIONAL DEFINITIONS ***************************
# These are various flags and settings that will be used by the
# compiler to generate code
#

#### ****** TESTING AND SECURITY LOOPHOLE ******************************
# Define TESTVERSION and GETENV to allow the LPD_CONFIG environment
#  variable to be used as the name of a configuration file.  In non-testing
#  systems,  this is a security loophole.
#TESTVERSION=yes
ifdef TESTVERSION
GETENV_CFLAGS = -DGETENV
endif

# the compiler optimisation/debugging flags you wish to use.
# These are in addition to the CCFLAGS and LDFLAGS
CCOPTFLAGS =
LDFLAGS    := -g $(LDFLAGS)

#********************** OVERRIDING CONFIGURES' GUESSES ******************
#
#
# If you get errors similar to the ones below, it may be
# because "configure" guessed wrong. You can override the
# guesses here, by uncommenting the appropriate line...

# "recvfiles.c: unknown struct fsb has no size":
# define STATFS: to use statfs(2) (BSD)
#        STATVFS: to use statvfs(2) (SVR4)
#     others for system specific cases
#
# STATFS_CFLAGS= -DMAKE_USE_STATFS=ULTRIX_STATFS
# STATFS_CFLAGS= -DMAKE_USE_STATFS=SVR3_STATFS
# STATFS_CFLAGS= -DMAKE_USE_STATFS=STATVFS
# STATFS_CFLAGS= -DMAKE_USE_STATFS=STATFS

# define MAKE_USE_STTY =
#    SGTTYB   to use struct sgttyb and <sgtty.h> (BSD)
#    TERMIO  to use struct termio and <termio.h> (old sysV)
#    TERMIOS to use struct termios and <termio.h> (SVR4)
```

```
#
# STTY_CFLAGS= -DMAKE_USE_STTY=SGTTYB
# STTY_CFLAGS= -DMAKE_USE_STTY=TERMIO
# STTY_CFLAGS= -DMAKE_USE_STTY=TERMIOS

# a better way to do this is to edit the "ARGH" section of portable.h,
# and add the appropriate lines to the section for your OS, or add a new
# section if one doesn't exist; then you can send me the patches and
# I'll incorporate them into the distribution.

#
# Try using this if you do not have rw_sockets or the rw_socket test failes
# RWSOCKETS_CFLAGS= -DUSE_RWSOCKETS

#
# You might discover that the default locking mechanism
# does not work.  Try one of the following to override the
# configured value: fcntl(), lockf(), flock(), and 0 to suppress locking
#
# LOCK_DEVS_CFLAGS=-DLOCK_DEVS=devlock_fcntl
# LOCK_DEVS_CFLAGS=-DLOCK_DEVS=devlock_lockf
# LOCK_DEVS_CFLAGS=-DLOCK_DEVS=devlock_flock
# LOCK_DEVS_CFLAGS=-DLOCK_DEVS=0
#

# SETPROCTITLE - overwrites the program argument information
#   when ps is used, displays status.  Used only by LPD
# if this does not work, define the following

# SETPROCTITLE_CFLAGS=-DNO_SETPROCTITLE

# Long or short LPQ status format
# if you want to have a 5-6 line status format,  then undefine the
# following:
# LONG_STATUS_FORMAT=-DLONG_STATUS_FORMAT

#######
#
#  Running filters with full root perms
# This is a security loophole - are you sure you want to do this?
# You can guess at my opinion by the name of the option.
#
# ROOT_CFLAG=-DROOT_PERMS_TO_FILTER_SECURITY_LOOPHOLE


#**********************  End of Configurable Part ****************************
#***************************** SYSTEM DEPENDENCIES ***************************

TARGET= \
                authenticate_pgp readfilecount removeoneline setupauth \
                liblpr.a checkpc lpr lpd lpq lprm lpc lpbanner \
                lpf lpraccnt monitor lp cancel lpstat
NONTARGET = sserver sclient testauth


SRCDIRS=${SRC}/common ${SRC}/LPD ${SRC}/LPQ ${SRC}/LPR \
        ${SRC}/LPRM ${SRC}/LPC ${SRC}/CHECKPC ${SRC}/LPBANNER ${SRC}/LPF \
        ${SRC}/LPRACCNT ${SRC}/MONITOR ${SRC}/AUTHENTICATE
INCLUDE=.. ${SRC}/include
```

```
EXTRA=${SRC}/TESTSUPPORT ${SRC}/UTILS

#GNU
VPATH=$(subst :, ,$(INCLUDE) $(SRCDIRS))
INCS= $(patsubst %,-I%,$(INCLUDE))
#
#BSD .PATH: $(INCLUDE) $(SRCDIRS)
#BSD INCS= $(INCLUDE:S/^/-I/g)

CFLAGS := $(CF) $(CCOPTFLAGS) -DHAVE_CONFIG_H $(INCS) \
        ${GETENV_CFLAGS} $(STTY_CFLAGS) $(STATFS_CFLAGS) $(RWSOCKETS_CFLAGS) \
        $(LOCK_DEVS_CFLAGS) $(SETPROCTITLE_CFLAGS) $(LONG_STATUS_FORMAT) \
        $(ROOT_CFLAG)

LIBLPR_OBJS = \
        bsd-compat.o checkremote.o cleantext.o controlword.o \
        copyright.o des.o data.o decodestatus.o default.o dump.o \
        errormsg.o fileopen.o fixcontrol.o freespace.o \
        getcnfginfo.o gethostinfo.o getopt.o getparms.o \
        getprinter.o getqueue.o getuserinfo.o globmatch.o inet_ntop.o \
        initialize.o jobcontrol.o killchild.o krb5_auth.o krb5_sendauth.o
link_support.o \
        lockfile.o malloclist.o merge.o messages.o parse_debug.o \
        pathname.o patselect.o permissions.o pr_support.o \
        printcap.o printjob.o proctitle.o protocolhandler.o \
        rand_key.o readstatus.o \
        removejob.o rw_pipe.o sendauth.o sendjob.o \
        sendlpc.o sendlprm.o sendlpq.o serverpid.o \
        setstatus.o setuid.o setup_filter.o setupprinter.o \
        snprintf.o spoolcontrol.o stty.o termclear.o \
        timeout.o utilities.o utils.o vars.o waitchild.o

LPD_OBJS= lpd.o \
        lpd_getparms.o lpd_rcvjob.o lpd_secure.o lpd_status.o lpd_jobs.o \
        lpd_sendmail.o lpd_remove.o lpd_control.o

LPR_OBJS= lpr.o lpr_getparms.o \
        lpr_chkparms.o lpr_cpyfiles.o lpr_makejob.o

LPQ_OBJS= lpq.o lpq_getparms.o

LPRM_OBJS= lprm.o lprm_getparm.o

LPC_OBJS= lpc.o lpc_getparms.o

CHECKPC_OBJS = checkpc.o checkpc_perm.o checkpc_port.o

LPBANNER_OBJS = lpbanner.o lpbanner_prn.o \
        lpbanner_fnt.o

LPF_OBJS = lpf.o

LPRACCNT_OBJS= lpraccnt.o

MONITOR_OBJS= monitor.o getopt.o

all: $(TARGET)

.PHONY: all ci clean \
```

```
        uninstall realclean mostlyclean distclean \
        TAGS info

# we want to force default.o to be updated whenever we
# change the options in Makefile or Makefile.bsd
#GNU
default.o: Makefile
#
#BSD default.o: Makefile.bsd

#
# use RANLIB with no ordering, or lorder/tsort if it is there
# if you do not have either, fall back and just build archive
# This appears to cover all cases for portability
#

liblpr.a: $(LIBLPR_OBJS)
        ar ruv $@ $(LIBLPR_ORDER)
        $(RANLIB) $@

lpr: $(LPR_OBJS) liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
lpd: $(LPD_OBJS) liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
lpq: $(LPQ_OBJS) liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
lprm: $(LPRM_OBJS) liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
lpc: $(LPC_OBJS) liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
checkpc: $(CHECKPC_OBJS) liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
lpbanner: $(LPBANNER_OBJS)
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
lpf: $(LPF_OBJS)
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
lpraccnt: $(LPRACCNT_OBJS) liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
monitor: $(MONITOR_OBJS)
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
readfilecount: readfilecount.o
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
removeoneline: removeoneline.o
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
setupauth: setupauth.o
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)

authenticate_pgp: authenticate_pgp.sh
        rm -f $@; cp $^ $@; chmod 555 $@

#ptest: common/snprintf.c
#       $(LDCC) $(CFLAGS) -DTEST -o ptest common/snprintf.c

sserver: sserver.o liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)
sclient: sclient.o liblpr.a
        $(LDCC) $(LDFLAGS) -o $@ $^ $(LIBS)

cancel:
```

```
	rm -f $@; ln -s lprm $@
lpstat:
	rm -f $@; ln -s lpq $@
lp:
	rm -f $@; ln -s lpr $@

##############################################################################
#
NORM_PERMS=	0755
SUID_ROOT_PERMS= 04755 -o root
PERMS=$(NORM_PERMS)
#PERMS=$(SUID_ROOT_PERMS)
##############################################################################

install: all ${INSTALL_LIB} ${INSTALL_BIN} ${INSTALL_MAINT}
	${INSTALLCMD} -m $(PERMS) lpq ${INSTALL_BIN}
	${INSTALLCMD} -m $(PERMS) lprm ${INSTALL_BIN}
	${INSTALLCMD} -m $(PERMS) lpr ${INSTALL_BIN}
	${INSTALLCMD} -m $(PERMS) lpc ${INSTALL_MAINT}
	${INSTALLCMD} -m $(NORM_PERMS) lpd ${INSTALL_LIB}
	${INSTALLCMD} -m $(NORM_PERMS) lpf ${INSTALL_LIB}
	${INSTALLCMD} -m $(NORM_PERMS) lpbanner ${INSTALL_LIB}
	${INSTALLCMD} -m $(NORM_PERMS) checkpc ${INSTALL_MAINT}
	${INSTALLCMD} -m $(NORM_PERMS) lpraccnt ${INSTALL_MAINT}
	${INSTALLCMD} -m $(NORM_PERMS) readfilecount ${INSTALL_LIB}
	${INSTALLCMD} -m $(NORM_PERMS) removeoneline ${INSTALL_LIB}
	${INSTALLCMD} -m $(NORM_PERMS) authenticate_pgp ${INSTALL_LIB}
	${INSTALLCMD} -m $(NORM_PERMS) setupauth ${INSTALL_LIB}
	rm -f ${INSTALL_BIN}/lp;     ln -s lpr  ${INSTALL_BIN}/lp;
	rm -f ${INSTALL_BIN}/lpstat; ln -s lpq  ${INSTALL_BIN}/lpstat;
	rm -f ${INSTALL_BIN}/cancel; ln -s lprm ${INSTALL_BIN}/cancel;

${INSTALL_LIB}::
	${SRC}/mkinstalldirs $@
${INSTALL_BIN}::
	${SRC}/mkinstalldirs $@
${INSTALL_MAINT}::
	${SRC}/mkinstalldirs $@


uninstall:
	rm -f ${INSTALL_LIB}/lpd
	rm -f ${INSTALL_LIB}/lpf
	rm -f ${INSTALL_LIB}/lpbanner
	rm -f ${INSTALL_BIN}/lpq
	rm -f ${INSTALL_BIN}/lprm
	rm -f ${INSTALL_BIN}/lpr
	rm -f ${INSTALL_MAINT}/lpc
	rm -f ${INSTALL_MAINT}/checkpc


##############################################################################


clean::
	-rm -f *.o *.core ? core $(TARGET) $(NONTARGET)

realclean mostlyclean distclean:: clean
	-rm -f Makefile Makefile.bsd tags
```

```
info:
tags TAGS:
        ctags -t -d common/*.c include/*.h LPC/*.c LPD/*.c \
                LPQ/*.c LPR/*.c LPRM/*.c

#############################################################################

cifiles:
        if [ ! -d RCS ] ; then mkdir RCS ; fi ;
        checkin() { \
                echo $$1 ; \
                ci $(CI) -l -mUpdate -t-Initial $$1; \
        }; \
        for i in *; do \
                if [ -f "$$i" ] ; then \
                        case $$i in \
                                Makefile*|*install* ) echo $$i; checkin $$i ;; \
                        esac; \
                fi; \
        done;

ci: cifiles
        checkin() { \
                echo $$1 ; \
                ci $(CI) -l -mUpdate -t-Initial $$1; \
        }; \
        for i in *; do \
                if [ -d "$$i" ] ; then \
                        case $$i in \
                        RCS ) ;; \
                        * ) if [ ! -d $$i/RCS ] ; then mkdir $$i/RCS ; fi ; \
                                for j in $$i/*; do \
                                        if [ -f "$$j" ] ; then \
                                                checkin $$j; \
                                        fi; \
                                done; \
                                ;; \
                        esac; \
                fi; \
        done;

depend: /tmp/dep /tmp/order
        ( sed -n '1,/^##.*GENERATED/p' ${SRC}/Makefile.in; \
                echo ; echo; \
        echo "########### LIBRARY ORDER #########"; \
        cat /tmp/order; \
        echo ; echo; \
        echo "########### DEPENDENCIES #########"; \
        cat /tmp/dep; ) >/tmp/Makefile.in
        cp ${SRC}/Makefile.in ${SRC}/Makefile.in.old
        cp /tmp/Makefile.in ${SRC}/Makefile.in


/tmp/dep: *.o
        ${SRC}/../UTILS/makeinc $^ >$@

# make the order in which library files should be loader
# this requires hand tuning the makefile, but it is easier than
```

```
# trying to fix up things with conditional tests
#
/tmp/order: $(LIBLPR_OBJS)
        lorder $^ |tsort >/tmp/raw
        awk '\
                BEGIN { print "LIBLPR_ORDER = \\"; } \
                { line = line " " $$0;            \
                        if( ++i >= 4 ){               \
                                print "\t" line "\\";     \
                                line = "";                \
                                i = 0;                    \
                        }                             \
                }                                 \
                END {                             \
                        if( line != "" ){             \
                                print "\t" line;          \
                        }                             \
                }' /tmp/raw >/tmp/order

#############################################################################

############## GENERATED ###################################################


############ LIBRARY ORDER #########
LIBLPR_ORDER = \
         termclear.o setupprinter.o serverpid.o sendlpq.o\
         sendlprm.o sendlpc.o sendjob.o sendauth.o\
         removejob.o readstatus.o proctitle.o printjob.o\
         pr_support.o protocolhandler.o des.o data.o messages.o\
         rand_key.o utils.o permissions.o patselect.o parse_debug.o\
         krb5_auth.o initialize.o getuserinfo.o getqueue.o\
         getprinter.o getparms.o getopt.o getcnfginfo.o\
         freespace.o fixcontrol.o controlword.o checkremote.o\
         stty.o spoolcontrol.o krb5_sendauth.o waitchild.o\
         utilities.o timeout.o setup_filter.o rw_pipe.o\
         printcap.o merge.o jobcontrol.o globmatch.o\
         default.o cleantext.o vars.o copyright.o\
         setuid.o setstatus.o link_support.o gethostinfo.o\
         dump.o inet_ntop.o malloclist.o lockfile.o\
         fileopen.o pathname.o killchild.o decodestatus.o\
         errormsg.o snprintf.o bsd-compat.o


############ DEPENDENCIES #########
bsd-compat.o :  config.h bsd-compat.h debug.h errormsg.h lp.h portable.h utili-
ties.h
checkpc.o :     config.h bsd-compat.h checkpc_perm.h debug.h dump.h errormsg.h
fileopen.h getcnfginfo.h getqueue.h initialize.h lp.h patchlevel.h pathname.h
permission.h portable.h printcap.h removejob.h setuid.h setupprinter.h timeout.h
utilities.h
checkpc_perm.o :        config.h bsd-compat.h checkpc_perm.h debug.h errormsg.h
lp.h pathname.h portable.h setuid.h utilities.h
checkpc_port.o :        config.h bsd-compat.h debug.h errormsg.h fileopen.h
freespace.h killchild.h lockfile.h lp.h pathname.h portable.h rw_pipe.h setuid.h
stty.h timeout.h utilities.h waitchild.h
checkremote.o : config.h bsd-compat.h checkremote.h debug.h errormsg.h geth-
ostinfo.h lp.h portable.h printcap.h utilities.h
cleantext.o :   config.h bsd-compat.h cleantext.h debug.h errormsg.h lp.h porta-
```

```
ble.h utilities.h
controlword.o : config.h bsd-compat.h control.h debug.h errormsg.h lp.h porta-
ble.h utilities.h
copyright.o :   config.h bsd-compat.h debug.h errormsg.h lp.h patchlevel.h porta-
ble.h utilities.h
des.o : shared.h protocol.h des.h reporting.h bsd-compat.h debug.h errormsg.h
lp.h
data.o : shared.h protocol.h reporting.h bsd-compat.h debug.h errormsg.h lp.h
decodestatus.o :       config.h bsd-compat.h debug.h decodestatus.h errorcodes.h
errormsg.h lp.h portable.h utilities.h
default.o :
dump.o :        config.h bsd-compat.h debug.h dump.h errormsg.h lp.h portable.h
utilities.h
errormsg.o :    config.h bsd-compat.h debug.h errormsg.h killchild.h lp.h porta-
ble.h setstatus.h utilities.h
fileopen.o :    config.h bsd-compat.h debug.h errorcodes.h errormsg.h fileopen.h
killchild.h lockfile.h lp.h pathname.h portable.h timeout.h utilities.h waitch-
ild.h
fixcontrol.o :  config.h bsd-compat.h cleantext.h debug.h dump.h errormsg.h fix-
control.h lp.h malloclist.h merge.h portable.h utilities.h
freespace.o :   config.h bsd-compat.h debug.h errormsg.h fileopen.h freespace.h
lp.h pathname.h portable.h utilities.h
getcnfginfo.o : config.h bsd-compat.h debug.h dump.h errormsg.h fileopen.h getc-
nfginfo.h lp.h malloclist.h merge.h portable.h printcap.h utilities.h
gethostinfo.o : config.h bsd-compat.h debug.h dump.h errormsg.h gethostinfo.h
lp.h malloclist.h portable.h utilities.h
getopt.o :      config.h bsd-compat.h debug.h errormsg.h lp.h portable.h utili-
ties.h
getparms.o :    config.h bsd-compat.h debug.h errormsg.h getparms.h lp.h porta-
ble.h utilities.h
getprinter.o :  config.h bsd-compat.h checkremote.h debug.h dump.h errormsg.h
getprinter.h lp.h portable.h printcap.h utilities.h
getqueue.o :    config.h bsd-compat.h cleantext.h debug.h dump.h errorcodes.h
errormsg.h fileopen.h gethostinfo.h getqueue.h globmatch.h jobcontrol.h lp.h
malloclist.h merge.h pathname.h permission.h portable.h utilities.h
getuserinfo.o : config.h bsd-compat.h debug.h errormsg.h getuserinfo.h lp.h por-
table.h setuid.h utilities.h
globmatch.o :   config.h bsd-compat.h debug.h errormsg.h globmatch.h lp.h porta-
ble.h utilities.h
inet_ntop.o :   config.h bsd-compat.h debug.h errormsg.h lp.h portable.h utili-
ties.h
initialize.o :  config.h bsd-compat.h debug.h dump.h errormsg.h fileopen.h getc-
nfginfo.h gethostinfo.h getuserinfo.h initialize.h killchild.h lp.h portable.h
printcap.h setuid.h utilities.h waitchild.h
jobcontrol.o :  config.h bsd-compat.h debug.h decodestatus.h dump.h errorcodes.h
errormsg.h jobcontrol.h lockfile.h lp.h malloclist.h pathname.h portable.h
pr_support.h setup_filter.h utilities.h
killchild.o :   config.h bsd-compat.h debug.h decodestatus.h errormsg.h kill-
child.h lp.h malloclist.h portable.h utilities.h
krb5_auth.o :   config.h bsd-compat.h debug.h errormsg.h fileopen.h killchild.h
krb5_auth.h lp.h portable.h utilities.h
krb5_sendauth.o :       config.h bsd-compat.h debug.h errormsg.h lp.h portable.h
utilities.h
link_support.o :        config.h bsd-compat.h debug.h errormsg.h gethostinfo.h
linksupport.h lp.h portable.h setuid.h timeout.h utilities.h
lockfile.o :    config.h bsd-compat.h debug.h errormsg.h fileopen.h lockfile.h
lp.h portable.h utilities.h
lpbanner.o :    lpbanner.h
lpbanner_fnt.o :        lpbanner.h
```

```
lpbanner_prn.o :        lpbanner.h
lpc.o : config.h bsd-compat.h control.h debug.h decodestatus.h errormsg.h get-
printer.h initialize.h killchild.h lp.h portable.h sendlpc.h setuid.h utili-
ties.h waitchild.h
lpc_getparms.o :        config.h bsd-compat.h debug.h errormsg.h lp.h
patchlevel.h portable.h utilities.h
lpd.o : config.h bsd-compat.h debug.h errormsg.h fileopen.h gethostinfo.h ini-
tialize.h killchild.h krb5_auth.h linksupport.h lp.h permission.h portable.h
printcap.h serverpid.h setstatus.h utilities.h waitchild.h
lpd_control.o : config.h bsd-compat.h cleantext.h control.h debug.h decodesta-
tus.h errormsg.h fileopen.h getqueue.h jobcontrol.h killchild.h linksupport.h
lp.h pathname.h patselect.h permission.h portable.h printcap.h serverpid.h set-
status.h setupprinter.h utilities.h waitchild.h
lpd_getparms.o :        config.h bsd-compat.h debug.h errormsg.h lp.h
patchlevel.h portable.h utilities.h
lpd_jobs.o :    config.h bsd-compat.h checkremote.h cleantext.h debug.h decod-
estatus.h dump.h errorcodes.h errormsg.h fileopen.h gethostinfo.h getqueue.h
jobcontrol.h killchild.h linksupport.h lockfile.h lp.h malloclist.h merge.h
pathname.h permission.h portable.h pr_support.h printcap.h printjob.h remove-
job.h sendjob.h serverpid.h setstatus.h setup_filter.h setupprinter.h utili-
ties.h waitchild.h
lpd_rcvjob.o :  config.h bsd-compat.h cleantext.h debug.h dump.h errorcodes.h
errormsg.h fileopen.h freespace.h gethostinfo.h getqueue.h jobcontrol.h kill-
child.h linksupport.h lockfile.h lp.h malloclist.h merge.h pathname.h permis-
sion.h portable.h serverpid.h setstatus.h setup_filter.h setupprinter.h
utilities.h
lpd_remove.o :  config.h bsd-compat.h checkremote.h cleantext.h debug.h decod-
estatus.h errorcodes.h errormsg.h gethostinfo.h jobcontrol.h killchild.h link-
support.h lp.h malloclist.h patselect.h permission.h portable.h pr_support.h
printcap.h removejob.h sendlprm.h setstatus.h setupprinter.h utilities.h waitch-
ild.h
lpd_secure.o :  config.h bsd-compat.h cleantext.h debug.h errorcodes.h
errormsg.h fileopen.h killchild.h krb5_auth.h linksupport.h lockfile.h lp.h mal-
loclist.h pathname.h permission.h portable.h setup_filter.h setupprinter.h util-
ities.h waitchild.h
lpd_sendmail.o :        config.h bsd-compat.h debug.h errorcodes.h errormsg.h
fileopen.h lp.h pathname.h portable.h pr_support.h setup_filter.h utilities.h
lpd_status.o :  config.h bsd-compat.h cleantext.h debug.h errormsg.h fileopen.h
gethostinfo.h getqueue.h jobcontrol.h killchild.h linksupport.h lp.h mallo-
clist.h pathname.h patselect.h permission.h portable.h printcap.h sendlpq.h
serverpid.h setup_filter.h setupprinter.h utilities.h
lpf.o :
lpq.o : config.h bsd-compat.h checkremote.h debug.h errormsg.h fileopen.h get-
printer.h initialize.h killchild.h lp.h malloclist.h portable.h printcap.h read-
status.h sendlpq.h termclear.h utilities.h
lpq_getparms.o :        config.h bsd-compat.h debug.h errormsg.h lp.h mallo-
clist.h patchlevel.h portable.h utilities.h
lpr.o : config.h bsd-compat.h debug.h dump.h errormsg.h initialize.h killchild.h
lp.h pathname.h portable.h sendjob.h setuid.h utilities.h
lpr_chkparms.o :        config.h bsd-compat.h debug.h errormsg.h getprinter.h
getuserinfo.h lp.h portable.h utilities.h
lpr_cpyfiles.o :        config.h bsd-compat.h debug.h errorcodes.h errormsg.h
fileopen.h killchild.h lp.h malloclist.h pathname.h portable.h utilities.h
lpr_getparms.o :       config.h bsd-compat.h debug.h errormsg.h getparms.h lp.h
patchlevel.h portable.h utilities.h
lpr_makejob.o : config.h bsd-compat.h debug.h dump.h errormsg.h lp.h mallo-
clist.h portable.h utilities.h
lpraccnt.o :    config.h bsd-compat.h debug.h errormsg.h lp.h portable.h utili-
ties.h
```

```
lprm.o :        config.h bsd-compat.h debug.h errormsg.h getprinter.h initial-
ize.h killchild.h lp.h portable.h sendlprm.h utilities.h
lprm_getparm.o :        config.h bsd-compat.h debug.h errormsg.h lp.h portable.h
printcap.h utilities.h
malloclist.o :  config.h bsd-compat.h debug.h errormsg.h lp.h malloclist.h por-
table.h utilities.h
merge.o :       config.h bsd-compat.h debug.h errormsg.h lp.h portable.h utili-
ties.h
messages.o : shared.h protocol.h reporting.h bsd-compat.h debug.h errormsg.h
lp.h
monitor.o :     config.h bsd-compat.h debug.h errormsg.h lp.h portable.h utili-
ties.h
parse_debug.o : config.h bsd-compat.h debug.h errorcodes.h errormsg.h kill-
child.h lp.h portable.h utilities.h
pathname.o :    config.h bsd-compat.h debug.h errormsg.h lp.h pathname.h porta-
ble.h utilities.h
patselect.o :   config.h bsd-compat.h debug.h errormsg.h globmatch.h lp.h patse-
lect.h portable.h utilities.h
permissions.o : config.h bsd-compat.h debug.h dump.h errormsg.h fileopen.h geth-
ostinfo.h globmatch.h lp.h malloclist.h permission.h portable.h setup_filter.h
utilities.h
pr_support.o :  config.h bsd-compat.h debug.h decodestatus.h errorcodes.h
errormsg.h fileopen.h killchild.h linksupport.h lockfile.h lp.h portable.h
pr_support.h printcap.h setstatus.h setup_filter.h stty.h timeout.h utilities.h
waitchild.h protocolhandler.h shared.h des.h protocol.h reporting.h
printcap.o :    config.h bsd-compat.h cleantext.h debug.h dump.h errormsg.h file-
open.h gethostinfo.h globmatch.h lp.h malloclist.h merge.h pathname.h portable.h
printcap.h setup_filter.h utilities.h
printjob.o :    config.h bsd-compat.h debug.h decodestatus.h dump.h errorcodes.h
errormsg.h fileopen.h jobcontrol.h killchild.h linksupport.h lp.h malloclist.h
portable.h pr_support.h printcap.h printjob.h setstatus.h setup_filter.h utili-
ties.h waitchild.h
proctitle.o :   config.h bsd-compat.h debug.h errormsg.h lp.h portable.h utili-
ties.h
protocolhander.o : protocolhandler.h shared.h protocol.h reporting.h bsd-com-
pat.h debug.h errormsg.h lp.h
rand_key.o : protocolhandler.h shared.h protocol.h reporting.h bsd-compat.h
debug.h errormsg.h lp.h
readfilecount.o :       config.h portable.h
readstatus.o :  config.h bsd-compat.h cleantext.h debug.h errormsg.h killchild.h
linksupport.h lp.h malloclist.h portable.h readstatus.h setstatus.h utilities.h
removejob.o :   config.h bsd-compat.h debug.h dump.h errormsg.h fileopen.h job-
control.h lp.h portable.h removejob.h setstatus.h utilities.h
removeoneline.o :       config.h portable.h
rw_pipe.o :     config.h bsd-compat.h debug.h errormsg.h lp.h portable.h
rw_pipe.h utilities.h
sclient.o :     config.h bsd-compat.h debug.h errormsg.h krb5_auth.h lp.h porta-
ble.h utilities.h
sendauth.o :    config.h bsd-compat.h cleantext.h debug.h errorcodes.h errormsg.h
fileopen.h killchild.h krb5_auth.h linksupport.h lp.h malloclist.h portable.h
printcap.h sendauth.h setstatus.h setup_filter.h utilities.h
sendjob.o :     config.h bsd-compat.h debug.h errorcodes.h errormsg.h fileopen.h
fixcontrol.h jobcontrol.h killchild.h linksupport.h lp.h malloclist.h portable.h
pr_support.h printcap.h printjob.h readstatus.h sendauth.h sendjob.h setstatus.h
setup_filter.h utilities.h
sendlpc.o :     config.h bsd-compat.h control.h debug.h errorcodes.h errormsg.h
fileopen.h getprinter.h killchild.h linksupport.h lp.h portable.h readstatus.h
sendauth.h sendlpc.h setstatus.h utilities.h
sendlpq.o :     config.h bsd-compat.h debug.h errormsg.h fileopen.h killchild.h
```

```
linksupport.h lp.h portable.h readstatus.h sendauth.h sendlpq.h utilities.h
sendlprm.o :    config.h bsd-compat.h debug.h errormsg.h fileopen.h killchild.h
linksupport.h lp.h portable.h readstatus.h sendauth.h sendlprm.h setstatus.h
utilities.h
serverpid.o :   config.h bsd-compat.h debug.h errormsg.h lp.h portable.h server-
pid.h utilities.h
setstatus.o :   config.h bsd-compat.h debug.h errorcodes.h errormsg.h fileopen.h
linksupport.h lp.h pathname.h portable.h setstatus.h utilities.h
setuid.o :     config.h bsd-compat.h debug.h errormsg.h lp.h portable.h setuid.h
utilities.h
setup_filter.o :       config.h bsd-compat.h cleantext.h debug.h decodestatus.h
dump.h errorcodes.h errormsg.h gethostinfo.h jobcontrol.h killchild.h lp.h mal-
loclist.h pathname.h portable.h pr_support.h printcap.h rw_pipe.h setstatus.h
setuid.h setup_filter.h timeout.h utilities.h waitchild.h
setupauth.o :   config.h portable.h
setupprinter.o :        config.h bsd-compat.h checkremote.h cleantext.h debug.h
dump.h errormsg.h fileopen.h getcnfginfo.h getqueue.h jobcontrol.h lp.h path-
name.h permission.h portable.h pr_support.h printcap.h setupprinter.h utili-
ties.h
snprintf.o :    config.h bsd-compat.h debug.h errormsg.h lp.h portable.h utili-
ties.h
spoolcontrol.o :        config.h bsd-compat.h debug.h errorcodes.h errormsg.h
lockfile.h lp.h malloclist.h pathname.h portable.h utilities.h
sserver.o :     config.h bsd-compat.h debug.h errormsg.h krb5_auth.h lp.h porta-
ble.h utilities.h
stty.o :        config.h bsd-compat.h debug.h errormsg.h lp.h portable.h stty.h
utilities.h
termclear.o :   config.h bsd-compat.h debug.h errormsg.h lp.h portable.h term-
clear.h utilities.h
timeout.o :     config.h bsd-compat.h debug.h errormsg.h lp.h portable.h time-
out.h utilities.h
utilities.o :   config.h bsd-compat.h debug.h errormsg.h lp.h portable.h time-
out.h utilities.h
utils.o : shared.h protocol.h reporting.h bsd-compat.h debug.h errormsg.h lp.h
vars.o :        config.h bsd-compat.h debug.h errormsg.h fileopen.h lp.h permis-
sion.h portable.h pr_support.h printcap.h setuid.h timeout.h utilities.h waitch-
ild.h
waitchild.o :   config.h bsd-compat.h debug.h decodestatus.h errormsg.h kill-
child.h lp.h portable.h timeout.h utilities.h waitchild.h
```

File Name: pr_support.c

```
/***************************************************************************
 * LPRng - An Extended Print Spooler System
 *
 * Copyright 1988-1997, Patrick Powell, San Diego, CA
 *     papowell@sdsu.edu
 * See LICENSE for conditions of use.
 *
 ***************************************************************************
 * MODULE: pr_support.c
 * PURPOSE: Printer opening
 ***************************************************************************/

static char *const _id =
"$Id: pr_support.c,v 3.10 1997/03/24 00:45:58 papowell Exp papowell $";

/***************************************************************************
 Commentary:
Patrick Powell, Fri May 19 08:01:34 PDT 1995
This code is taken almost directly from the PLp_device Version 4.0 code.
Additional sanity checks and cleanups have been added.

 ***************************************************************************
 * MODULE: Print_support.c
 * handle the actual output to the Printer
 ***************************************************************************
 * int Print_open( int timeout ): opens the Printer
 * int Print_close(): closes the Printer
 * int Print_ready(): combines Print_open() and Print_of_fd()
 * int of_stop(): stops the 'of' filter
 * int Print_string( str, len ) : prints a string through 'of' or to Printer
 * int Print_copy( fd) : copies a file through 'of' or to Printer
 * int Print_filter( file, cmd) : makes a filter and copies file
 * int Print_banner(): prints a banner through 'of' or to Printer
 * Note: the above procedures which return values return JSUCC on success,
 *   and JFAIL or JABORT on failure
 ***************************************************************************/

#include "protocolhandler.h"
#include "lp.h"
#include "printcap.h"
#include "decodestatus.h"
#include "errorcodes.h"
#include "fileopen.h"
#include "killchild.h"
#include "linksupport.h"
#include "lockfile.h"
#include "pr_support.h"
#include "setstatus.h"
#include "setup_filter.h"
#include "stty.h"
#include "timeout.h"
#include "waitchild.h"
#include "rw_pipe.h"
/**** ENDINCLUDE ****/

/***************************************************************************
```

```
 * int Print_open( struct filter *filter, struct control_file *cfp,
 *   int timeout, int interval, int max_try )
 *
 *      Open the Printer, and set the filter structure with the pid and output
 *   device.
 *      If the RW printcap flag is set, output is opened RW, otherwise
 *      opened writeonly in append mode.
 *
 * Side Effect:
 *   terminates if Printer is unable to be opened
 ************************************************************************/

int Print_open( struct filter *filter,
        struct control_file *cfp, int timeout, int interval, int grace,
       int max_try, struct printcap_entry *printcap_entry, int accounting_port )
{
        int attempt = 0, err = 0;
        struct stat statb;
        int device_fd;
        time_t tm;
        char tm_str[32];
        char *s, *end;

        DEBUG3( "Print_open: device '%s', max_try %d, interval %d, timeout %d",
                Lp_device, max_try, interval, timeout );
        time( &tm );
        tm_str[0] = 0;
        safestrncat( tm_str, Time_str(1,tm) );
        if( Lp_device == 0 || *Lp_device == 0 ){
               fatal(LOG_ERR, "Print_open: printer '%s' missing Lp_device value",
                        Printer );
        }
        if( Lp_device[0] != '|' && (s = strchr( Lp_device, '%' )) ){
                /* we have a host%port form */
                *s++ = 0;
                end = s;
                Destination_port = strtol( s, &end, 10 );
                if( ((end - s) != strlen( s )) || Destination_port < 0 ){
                fatal( LOG_ERR,
                        "Print_open: 'lp' entry has bad port number '%s'",
                                Lp_device );
                }
        } else if( Lp_device[0] != '|' && Lp_device[0] != '/' ){
                fatal(LOG_ERR, "Print_open: printer '%s', bad 'lp' entry '%s'",
                        Printer, Lp_device );
        }

        if (grace) {
                setstatus( cfp, "Waiting Grace period '%s' : '%d'",
                        Lp_device, grace);
                plp_sleep(grace);
        }

        /* set flag for closing on timeout */
        do{
                setstatus( cfp,
                "opening '%s' at %s, attempt %d",
                        Lp_device, tm_str, ++attempt);
```

```
                filter->pid = 0;
                filter->input = 0;
                switch( Lp_device[0] ){
                case '|':
                        if( DevNullFD <= 0 ){
                             logerr_die( LOG_CRIT, "Print_open: bad DevNullFD");
                        }
                        if( Make_filter( 'f', cfp, filter,
                                Lp_device+1, 1, Read_write, DevNullFD,
printcap_entry, (void *)0,
                                accounting_port, Logger_destination != 0, 0 ) ){
                                setstatus( cfp, "%s", cfp->error );
                                filter->input = -1;
                        }
                        device_fd = filter->input;
                        break;
                case '/':
                        device_fd = -1;
                        device_fd = Checkwrite_timeout( timeout, Lp_device,
&statb,
                                        (Lock_it || Read_write)
                                              ?(O_RDWR):(O_APPEND|O_WRONLY), 0,
Nonblocking_open );
                        err = errno;

                        /* Note: there is no timeout race condition here -
                                we either set the device_fd or we did not */
                        if( Lock_it && device_fd >= 0 ){
                                /*
                                 * lock the device so that multiple servers can
                                 * use it
                                 */
                                if( isatty( device_fd ) ){
                                      if( LockDevice( device_fd, Lp_device ) <=
0 ){
                                              err = errno;
                                              setstatus( cfp,
                                                    "Device '%s' is locked!",
Lp_device );
                                              close( device_fd );
                                              device_fd = -1;
                                       }
                                } else if( S_ISREG(statb.st_mode ) ){
                                      if( Do_lock( device_fd, Lp_device, 1 ) <=
0 ){
                                              err = errno;
                                              setstatus( cfp,
                                                    "Device '%s' is locked!",
Lp_device );
                                              close( device_fd );
                                              device_fd = -1;
                                       }
                                }
                        }
                        filter->input = device_fd;
                        break;
                default:
                        device_fd = Link_open( Lp_device, max_try, timeout);
                        err = errno;
```

```
                filter->input = device_fd;
                break;
        }

        if( device_fd < 0 )
          {
            errno = err;
            DEBUG3( "Print_open: cannot open device '%s' - '%s'",
                    Lp_device, Errormsg(err) );
            if( (max_try == 0 || attempt < max_try) && interval > 0 )
              {
                setstatus( cfp, "Cannot open '%s' - '%s', sleeping %d",
                           Lp_device, Errormsg( err), interval );
                plp_sleep( interval );
              }
            else
              {
                break;
              }
          }
} while( device_fd < 0 );

if( device_fd < 0 ){
        setstatus( cfp, "cannot open '%s' after %d attempts - %s",
                Lp_device, attempt, Errormsg(err) );
        Errorcode = JFAIL;
} else if( isatty( device_fd ) ){
        /*
         * if it is a tty, set the baud rates and control information;
         * Note that the open O_NCTTY prevents this from being a control
         *  device for the process.
         */
        Do_stty( device_fd );
}

if(Is_Mini_Supervisor)
  {
    int printerFD;
    int fds[2];
    int status;

    fds[0] = fds[1] = -1;
    printerFD = device_fd;

    status = pipe(fds);
    if(status < 0)
      fatal(LOG_ERR, "Print_open: pipe() failed - %s",
            Errormsg(errno));

    status = dofork();

    if(status < 0)
      fatal(LOG_ERR, "Print_open: dofork() failed - %s",
            Errormsg(errno));

    if(status == 0)
      {
        /* We are the child */
        close(fds[0]);   /* only the parent should have this */
```

```
                ProtocolHandler(fds[1], printerFD, Mini_Supervisor_Encrypt);
                close(printerFD);
                exit(0);
              }

            close(fds[1]);       /* only the child should have this */
            device_fd = fds[0];  /* where the filter talks to */
          }

        filter->input = device_fd;
        DEBUG("Print_open: %s is fd %d", Lp_device, device_fd);

        return( device_fd );
}


/*************************************************************************
 * Print_flush()
 * 1. Flush all of the open pipe and/or descriptors
 *************************************************************************/
void Print_flush( void )
{
        DEBUG3("Print_flush: flushing printers");
        Flush_filter( &Device_fd_info );
        Flush_filter( &OF_fd_info );
        Flush_filter( &XF_fd_info );
        Flush_filter( &Pr_fd_info );
        Flush_filter( &Af_fd_info );
        Flush_filter( &As_fd_info );
}

/*************************************************************************
 * Print_close( void )
 * 1. Close all of the open pipe and/or descriptors
 * 2. Signal the filter processes to start up
 * 3. We wait for a while for them to complete
 *************************************************************************/

void Print_close(int timeout)
{
        DEBUG3("Print_close: closing printers");
        Print_flush();
        /* shut them all down first */
        Close_filter( &Device_fd_info, timeout, "Print_close" );
        Close_filter( &OF_fd_info, timeout, "Print_close" );
        Close_filter( &XF_fd_info, timeout, "Print_close" );
        Close_filter( &Pr_fd_info, timeout, "Print_close" );
        Close_filter( &Af_fd_info, timeout, "Print_close" );
        Close_filter( &As_fd_info, timeout, "Print_close" );
}

/*************************************************************************
 * Print_kill( signal )
 * 1. Flush all of the open pipe and/or descriptors
 * 2. Close all of the open pipe and/or descriptors
 * 3. Kill them off if necessary
 *************************************************************************/
```

```
void Print_kill( int signal )
{
        DEBUG3("Print_kill: killing filters");
        Print_flush();
        Kill_filter( &Device_fd_info, signal );
        Kill_filter( &OF_fd_info, signal );
        Kill_filter( &XF_fd_info, signal );
        Kill_filter( &Pr_fd_info, signal );
        Kill_filter( &Af_fd_info, signal );
        Kill_filter( &As_fd_info, signal );
}

/***************************************************************************
 * Print_abort()
 * 1. Close all of the open pipe and/or descriptors
 * 2. Wait for all the filter processes to terminate
 * 3. After timeout period, kill them off
 *
 ***************************************************************************/

void Print_abort( void )
{
        pid_t result;
        plp_status_t status;
        int step = 0;
        int err;

        DEBUG1( "Print_abort: starting shutdown" );

        /* be gentle at first */
        Print_close(-1);
        while(1){
                DEBUG1( "Print_abort: gathering children" );
                do{
                        status = 0;
                        result = plp_waitpid_timeout( 10, -1, &status, 0 );
                        err = errno;
                        DEBUG1( "Print_abort: result %d, status 0x%x", result,
status );
                } while( result > 0 );
                if( (result == -1 ) && err == ECHILD ){
                        break;
                }
                DEBUG4( "Print_abort: step %d, killing", step, status );
                switch( step++ ){
                       case 0: Print_kill( SIGINT ); break;     /* hit it once */
                      case 1: Print_kill( SIGQUIT ); break;   /* hit it harder */
                        case 2: Print_kill( SIGKILL ); break;   /* hit it really
hard */
                        case 3: return; /* give up */
                }
        }
        DEBUG1( "Print_abort: done" );
}

int of_start( struct filter *filter )
{
       DEBUG1 ("of_start: sending SIGCONT to OF_Filter (pid %d)", filter->pid );
        if( filter->pid > 0 && kill( filter->pid, SIGCONT) < 0 ){
```

```
                logerr( LOG_WARNING, "cannot restart output filter");
                return( JFAIL);
        }
        return JSUCC;
}

/*************************************************************************
 * of_stop()
 * wait for the output filter to stop itself
 * Return: JSUCC if stopped, JFAIL if it died
 *************************************************************************/

static char filter_stop[] = "\031\001";         /* magic string to stop OF_Filter
*/

int of_stop( struct filter *filter, int timeout )
{
        int pid = 0;
        int err;
        plp_status_t statb;
        DEBUG2 ("of_stop: writing stop string to fd %d, pid %d",
                filter->input, filter->pid );
        if( Print_string( filter, filter_stop, strlen( filter_stop ),
                        timeout ) != JSUCC ){
                return( JFAIL );
        }

        DEBUG2 ("of_stop: waiting for OF_Filter %d", filter->pid );
        statb = 0;
        do{
              pid = plp_waitpid_timeout(timeout, filter->pid, &statb, WUNTRACED
);
                err = errno;
                DEBUG3( "of_stop: pid %d, statb 0x%x", pid, statb );
        }while( Alarm_timed_out == 0 && pid == -1 && err != ECHILD );

        if( Alarm_timed_out || !WIFSTOPPED (statb) ){
                logerr( LOG_WARNING, "of_stop: did not stop or died (%s)",
                                Decode_status( &statb));
                return( JFAIL);
        }
        DEBUG2 ("of_stop: output filter stopped");

        return( JSUCC);
}


/*************************************************************************
 * Print_string( struct filter, char *str, int len, int timeout )
 * print a string through a filter
 * 1. Enable the line Printer
 * 2. get the filter or device fd;
 * 3. put out the string
 * 4. if unsuccessful, close the Printer
 * Return: JSUCC if successful, JFAIL otherwise
 *************************************************************************/

int Print_string( struct filter *filter, char *str, int len, int timeout )
{
```

```
        int i = 0;              /* ACME again */
        int err;

        if( str == 0 || len <= 0 ){
                return( JSUCC);
        }
        i = Write_fd_len_timeout( timeout, filter->input, str, len );
        err = errno;
        if( Alarm_timed_out || i < 0 ){
                if( Alarm_timed_out ){
                        log( LOG_WARNING, "Print_string: write timeout error");
                } else {
                        logerr( LOG_WARNING, "Print_string: write error - %s",
                                Errormsg(err) );
                }
                return( JFAIL);
        }
        return( JSUCC);
}


/*************************************************************************
 * Print_copy( int fd, struct filter *filter, int timeout )
 * copy a file through a filter
 * 1. copy the file to the appropriate output device
 * 2. if an error, close the Printer
 *************************************************************************/

int Print_copy( struct control_file *cfp, int fd, struct stat *statb,
        struct filter *filter, int timeout, int status, char *file_name )
{
        char buf[LARGEBUFFER];
        int in;                     /* bytes read */
        int t = 0;                                      /* amount written */
        int total = 0;
        int err;                                        /* errno status */
        int oldfraction;                        /* how much done */
        int quanta = 4;                         /* resolution of amount */

       DEBUG3("Print_copy: fd %d to fd %d, pid %d, timeout %d, size %d, file_name
%s",
                fd, filter->input, filter->pid, timeout, (int)(statb->st_size),
file_name );


        oldfraction = 0;
        while( (in = read( fd, buf, sizeof( buf) ) ) > 0 ){
                DEBUG4("Print_copy: read %d, total %d", in, total );
                t = Write_fd_len_timeout( timeout, filter->input, buf, in );
                err = errno;

                DEBUG3("Print_copy: printed %d", t );
                if( Alarm_timed_out || t < 0 ){
                        if( Alarm_timed_out ){
                                if( status ){
                                        setstatus( cfp, "timeout after writing %d
bytes",
                                                total );
                                }
```

```
                                logerr( LOG_WARNING,
                                        "Print_copy: write error timeout after %d
bytes");
                        } else {
                                if( status ){
                                     setstatus( cfp, "write error after %d bytes
'%s'",
                                          total, Errormsg(err) );
                                }
                                errno = err;
                               logerr( LOG_WARNING, "write error after %d bytes",
total );
                        }
                        return( JFAIL);
                }
                total += in;
                t = (int)(quanta*((double)(total)/statb->st_size));
                if( t != oldfraction ){
                        oldfraction = t;
                        t = (100*t)/quanta;
                        setstatus( cfp, "printed %d percent of %d bytes of %s",
                                t,(int)(statb->st_size), file_name);
                }
        }
        /*
         * completed the reading
         */
        err = errno;
        if( in < 0 ){
                if( status ){
                        setstatus( cfp, "read error after '%d' bytes, '%s'",
                        total, Errormsg(err) );
                }
                errno = err;
                logerr( LOG_WARNING, "Print_copy: read error");
                return( JFAIL);
        }
        if( status )setstatus( cfp, "printed all %d bytes", total );

        DEBUG2("Print_copy: printed %d bytes", total);
        return( JSUCC);
}
```

File Name: protocolhandler.c

```
#include "protocolhandler.h"


void ProtocolHandler(int parentFD, int miniFD, int encrypt)
{
  ssize_t     bytesRead, bytesWritten;
  char        buffer[MAXLEN];
  int         gotBeginJob = 0, gotSendKey = 0, gotParentEOF = 0, gotMiniEOF = 0;
  char        jobDataBuffer[MAX_JOB_LEN];
  char        responseDataBuffer[MAX_RESPONSE_LEN];
  long        jobDataBytes;
  long        responseDataBytes;
  int         theError = NO_ERR;
  Keys        jobKeys, responseKeys;
  int         responseBufferFull = 0;
  int         timeOut = 2, maxRetries = 5;


  /* Loop on trying to get a begin job from the supervisor */
  while(!gotBeginJob && !gotSendKey)
    {
      if(encrypt)
        sprintf(buffer, "%c\n", START_ENCRYPTED_JOB_MSG);
      else
        sprintf(buffer, "%c\n", START_UNENCRYPTED_JOB_MSG);

      bytesWritten = WriteMessage(miniFD, buffer, strlen(buffer));
      if(bytesWritten < strlen(buffer))
        break;

      bytesRead = ReadMessage(miniFD, buffer);
      if(bytesRead <= 0)
        {
          ERROR("ProtocolHandler: supervisor had closed the connection.");
          close(miniFD);
          exit(1);
        }

      switch(buffer[0])
        {
        case NO_UNENCRYPTED_JOB_MSG:
          ERROR("ProtocolHandler: mini-supervisor refused clear text job.");
          ERROR("ProtocolHandler: switching to encrypted mode.");
          encrypt = 1;

          close(miniFD);

          /* now reopen the connection and try again */
          miniFD = Link_open( Lp_device, maxRetries, timeOut);
          theError = errno;

          if(miniFD < 0)
            {
              ERROR("ProtocolHandler: failed to reopen a connection: %s",
                    Errormsg(theError));
              exit(1);
```

```
        }
      break;

    case FATAL_ERROR_MSG:
      ERROR("ProtocolHandler: received 'fatal error' message.");
      exit(1);

    case BUSY_MSG:
      DEBUG("ProtocolHandler: mini-supervisor is busy. Sleeping.");
      close(miniFD);
      sleep(5);

      /* now reopen the connection and try again */
      miniFD = Link_open( Lp_device, maxRetries, timeOut);
      theError = errno;

      if(miniFD < 0)
        {
          ERROR("ProtocolHandler: failed to reopen a connection: %s",
                Errormsg(theError));
          exit(1);
        }
      break;

    case BEGIN_JOB_MSG:
      if(!encrypt)
        gotBeginJob = 1;
      else
        {
          ERROR("ProtocolHandler: begin job received for encrypted job.");
          SendFatalError(miniFD);
          exit(1);
        }
      break;

    case SEND_KEY_MSG:
      if(encrypt)
        gotSendKey = 1;
      else
        {
          ERROR("ProtocolHandler: send key received for unencrypted job.");
          SendFatalError(miniFD);
          exit(1);
        }
      break;

    default:
      ERROR("ProtocolHandler: invalid reply to job request message.");
      SendFatalError(miniFD);
      exit(1);
    }
  }

/* establish keys with the mini supervisor if we are encrypting */
if(gotSendKey)
  {
    FourBytes  jobNumber = 0;
    int        numArgsFilled;
```

```
    /* get the amount of data that will be sent */
    numArgsFilled = sscanf(&(buffer[1]), "%d\n", &jobNumber);

    if(numArgsFilled != 1)
      {
        ERROR("ProtocolHandler: invalid send key message.");
        SendFatalError(miniFD);
        exit(1);
      }

    theError = EstablishKeys(miniFD, jobNumber, &jobKeys, &responseKeys);

    if(theError != NO_ERR)
      exit(1);
  }

jobDataBytes = 0;
responseDataBytes = 0;
responseBufferFull = 0;

while(!(gotMiniEOF ||
        (gotParentEOF && jobDataBytes == 0 && responseDataBytes == 0)))
  {
    fd_set readfds, writefds;
    int readyConnections;

    FD_ZERO(&readfds);
    FD_ZERO(&writefds);

    /* check for data to read if we have space */
    if(jobDataBytes < MAX_JOB_LEN && !gotParentEOF)
      FD_SET(parentFD, &readfds);
    if(responseDataBytes < MAX_RESPONSE_LEN && !gotMiniEOF)
      FD_SET(miniFD, &readfds);

    /* If we have data to write, check for that */
    if(jobDataBytes > 0)
      FD_SET(miniFD, &writefds);
    if(responseDataBytes > 0)
      FD_SET(parentFD, &writefds);

    readyConnections = select(maximum(miniFD, parentFD) + 1,
                              &readfds, &writefds, 0, 0);
    theError = errno;

    if(readyConnections < 0)
      {
        if(theError != EINTR)
          {
            ERROR("ProtocolHandler: select() failed: %s", Errormsg(theError));
            exit(1);
          }
        continue;   /* try again */
      }

    if(readyConnections == 0)
      continue;
```

```
/* can we read a data messsage from the parent? */
if(jobDataBytes < MAX_JOB_LEN &&
   FD_ISSET(parentFD, &readfds))
  {
    DEBUG("ProtocolHandler: reading data from the parent.");

    theError = GetData(parentFD,
                       jobDataBuffer,
                       &jobDataBytes,
                       MAX_JOB_LEN);

    if(theError == GOT_EOF)
      gotParentEOF = 1;

    if(theError == FATAL)
      exit(1);

    continue;
  }

/* can we read data from the mini-supervisor? */
if(responseDataBytes < MAX_RESPONSE_LEN &&
   !responseBufferFull &&
   FD_ISSET(miniFD, &readfds))
  {
    DEBUG("ProtocolHandler: reading data from the mini-supervisor.");

    if(encrypt)
      theError = EGetData(miniFD,
                          responseDataBuffer,
                          &responseDataBytes,
                          MAX_RESPONSE_LEN,
                          &responseKeys);
    else
      theError = GetData(miniFD,
                         responseDataBuffer,
                         &responseDataBytes,
                         MAX_RESPONSE_LEN);

    if(theError == GOT_EOF)
      gotMiniEOF = 1;

    if(theError == FATAL)
      exit(1);

    if(theError == FULL)
      responseBufferFull = 1;

    continue;
  }

/* can we write data to the mini-supervisor? */
if(jobDataBytes > 0 && FD_ISSET(miniFD, &writefds))
  {
    DEBUG("ProtocolHandler: writing data to the mini-supervisor.");

    if(encrypt)
      theError = ESendData(miniFD,
                           jobDataBuffer,
```

```
                                &jobDataBytes,
                                MAX_JOB_LEN,
                                &jobKeys);
          else
            theError = SendData(miniFD,
                                jobDataBuffer,
                                &jobDataBytes,
                                MAX_JOB_LEN);

          if(theError != NO_ERR)
            exit(1);
        }


      /* can we write a response message to our parent? */
      if(responseDataBytes > 0 && FD_ISSET(parentFD, &writefds))
        {
          DEBUG("ProtocolHandler: writing data to the parent.");

          theError = SendData(parentFD,
                              responseDataBuffer,
                              &responseDataBytes,
                              MAX_RESPONSE_LEN);

          if(theError != NO_ERR)
            exit(1);

          responseBufferFull = 0;
        }
    }

  if(jobDataBytes != 0 || responseDataBytes != 0)
    ERROR("We failed to send something.");

  /* zero out the keys to be safe */
  ZeroKeys(jobKeys);
  ZeroKeys(responseKeys);

  close(miniFD);
  return;
}



int EstablishKeys(int miniFD, FourBytes jobNumber,
                  Keys *jobKeys, Keys *responseKeys)
{
  char               keyBuffer[MAX_KEY_LEN];
  unsigned long      keyDataSize;
  unsigned long      tempSize;
  char               buffer[MAXLEN];
  Keys               masterKeys;
  des_cblock         jk1, jk2, jk3, rk1, rk2, rk3;
  int                theError;
  ssize_t            bytesRead, bytesWritten;
  FourBytes          tempNumber;

  /* generate the keys */
  des_random_key(jk1);
```

```
des_random_key(jk2);
des_random_key(jk3);
des_random_key(rk1);
des_random_key(rk2);
des_random_key(rk3);

/* put the keys and job number in the buffer */
keyDataSize = 0;
memcpy(&keyBuffer[keyDataSize], jk1, sizeof(des_cblock));
keyDataSize += sizeof(des_cblock);
memcpy(&keyBuffer[keyDataSize], jk2, sizeof(des_cblock));
keyDataSize += sizeof(des_cblock);
memcpy(&keyBuffer[keyDataSize], jk3, sizeof(des_cblock));
keyDataSize += sizeof(des_cblock);
memcpy(&keyBuffer[keyDataSize], rk1, sizeof(des_cblock));
keyDataSize += sizeof(des_cblock);
memcpy(&keyBuffer[keyDataSize], rk2, sizeof(des_cblock));
keyDataSize += sizeof(des_cblock);
memcpy(&keyBuffer[keyDataSize], rk3, sizeof(des_cblock));
keyDataSize += sizeof(des_cblock);

tempNumber = htonl(jobNumber);
memcpy(&keyBuffer[keyDataSize], &tempNumber, sizeof(FourBytes));
keyDataSize += sizeof(FourBytes);

/* write out the key message */
sprintf(buffer, "%c\n", KEY_MSG);
bytesWritten = WriteMessage(miniFD, buffer, strlen(buffer));
if(bytesWritten <= 0)
  {
    ERROR("EstablishKeys: failed writing key message.");
    return(FATAL);
  }

/* Get the master keys */
theError = GetMasterKeys(&masterKeys);
if(theError != NO_ERR)
  return(FATAL);

/* Send the data */
tempSize = keyDataSize;
theError = ESendData(miniFD, keyBuffer, &tempSize, MAX_KEY_LEN, &masterKeys);

/* zero out the master keys and buffer now that we are done with them */
ZeroKeys(masterKeys);
memset(keyBuffer, 0, keyDataSize);

if(theError != NO_ERR || tempSize != 0)
  return(FATAL);

/* Try to get the begin job message */
bytesRead = ReadMessage(miniFD, buffer);
if(bytesRead <= 0)
  {
    ERROR("EstablishKeys: supervisor had closed the connection.");
    close(miniFD);
    exit(1);
  }
```

```
  switch(buffer[0])
    {
    case FATAL_ERROR_MSG:
      ERROR("EstablishKeys: received 'fatal error' message.");
      return(FATAL);

    case JOB_NUMBER_MISMATCH_MSG:
      ERROR("EstablishKeys: received job number mismatch message");
      return(FATAL);

    case BEGIN_JOB_MSG:
      DEBUG("EstablishKeys: got begin job message.");
      break;

    default:
      ERROR("EstablishKeys: invalid reply to key message.");
      SendFatalError(miniFD);
      return(FATAL);
    }

  /* generate the key schedules */
  des_set_key(jk1, &jobKeys->key1);
  des_set_key(jk2, &jobKeys->key2);
  des_set_key(jk3, &jobKeys->key3);
  des_set_key(rk1, &responseKeys->key1);
  des_set_key(rk2, &responseKeys->key2);
  des_set_key(rk3, &responseKeys->key3);

  /* initialize the state vectors */
  memset(jobKeys->ivec1, 0, sizeof(des_cblock));
  memset(jobKeys->ivec2, 0, sizeof(des_cblock));
  memset(jobKeys->ivec3, 0, sizeof(des_cblock));
  memset(responseKeys->ivec1, 0, sizeof(des_cblock));
  memset(responseKeys->ivec2, 0, sizeof(des_cblock));
  memset(responseKeys->ivec3, 0, sizeof(des_cblock));

  /* zero out the temporary keys */
  memset(jk1, 0, sizeof(des_cblock));
  memset(jk2, 0, sizeof(des_cblock));
  memset(jk3, 0, sizeof(des_cblock));
  memset(rk1, 0, sizeof(des_cblock));
  memset(rk2, 0, sizeof(des_cblock));
  memset(rk3, 0, sizeof(des_cblock));

  return(NO_ERR);
}
```

File Name: protocolhandler.h

```
#include "shared.h"
#include "lp.h"
#include "linksupport.h"

/* function prototypes */
void ProtocolHandler(int parentFD, int miniFD, int encrypt);
int EstablishKeys(int miniFD, FourBytes jobNumber,
                  Keys *jobKeys, Keys *responseKeys);

/* rand_key.c */
void des_random_key(unsigned char *ret);
void des_set_odd_parity(des_cblock *key);
unsigned long des_cbc_cksum(des_cblock *input, des_cblock *output,
                            long length, DESContext *schedule,
                            des_cblock *ivec);
```

File Name: rand_key.c

```
/* The following source was taken from libdes by Eric Young. */

/* Copyright (C) 1995-1997 Eric Young (eay@mincom.oz.au)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@mincom.oz.au).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@mincom.oz.au).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * 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 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 cryptographic software written by
 *     Eric Young (eay@mincom.oz.au)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@mincom.oz.au)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``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 AUTHOR 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.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
```

```
 * [including the GNU Public Licence.]
 */

#include "protocolhandler.h"

#define l2c(l,c)        (*((c)++)=(unsigned char)(((l)     )&0xff), \
                         *((c)++)=(unsigned char)(((l)>> 8L)&0xff), \
                         *((c)++)=(unsigned char)(((l)>>16L)&0xff), \
                         *((c)++)=(unsigned char)(((l)>>24L)&0xff))

#define c2l(c,l)        (l =((unsigned long)(*((c)++)))    , \
                         l|=((unsigned long)(*((c)++)))<< 8L, \
                         l|=((unsigned long)(*((c)++)))<<16L, \
                         l|=((unsigned long)(*((c)++)))<<24L)

/* NOTE - c is not incremented as per c2l */
#define c2ln(c,l1,l2,n) { \
                        c+=n; \
                        l1=l2=0; \
                        switch (n) { \
                        case 8: l2 =((unsigned long)(*(--(c))))<<24L; \
                        case 7: l2|=((unsigned long)(*(--(c))))<<16L; \
                        case 6: l2|=((unsigned long)(*(--(c))))<< 8L; \
                        case 5: l2|=((unsigned long)(*(--(c)))); \
                        case 4: l1 =((unsigned long)(*(--(c))))<<24L; \
                        case 3: l1|=((unsigned long)(*(--(c))))<<16L; \
                        case 2: l1|=((unsigned long)(*(--(c))))<< 8L; \
                        case 1: l1|=((unsigned long)(*(--(c)))); \
                                } \
                        }


void des_random_key(unsigned char *ret)
{
  DESContext ks;
  static unsigned long c=0;
  static unsigned short pid=0;
  static des_cblock data={0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef};
  des_cblock key;
  unsigned char *p;
  unsigned long t;

  if (!pid) pid=getpid();

  p=key;
  t=(unsigned long)time(NULL);
  l2c(t,p);
  t=(unsigned long)((pid)|((c++)<<16));
  l2c(t,p);

  des_set_odd_parity((des_cblock *)data);
  des_set_key(data, &ks);
  des_cbc_cksum((des_cblock *)key, (des_cblock *)key,
                (long)sizeof(key), &ks, (des_cblock *)data);

  des_set_odd_parity((des_cblock *)key);
  des_set_key(key, &ks);
  des_cbc_cksum((des_cblock *)key, (des_cblock *)data,
                (long)sizeof(key), &ks, (des_cblock *)key);
```

```
  memcpy(ret,data,sizeof(key));
  memset(key,0,sizeof(key));
  memset(&ks,0,sizeof(ks));
  t=0;
}

static const unsigned char odd_parity[256]={
  1,  1,  2,  2,  4,  4,  7,  7,  8,  8, 11, 11, 13, 13, 14, 14,
 16, 16, 19, 19, 21, 21, 22, 22, 25, 25, 26, 26, 28, 28, 31, 31,
 32, 32, 35, 35, 37, 37, 38, 38, 41, 41, 42, 42, 44, 44, 47, 47,
 49, 49, 50, 50, 52, 52, 55, 55, 56, 56, 59, 59, 61, 61, 62, 62,
 64, 64, 67, 67, 69, 69, 70, 70, 73, 73, 74, 74, 76, 76, 79, 79,
 81, 81, 82, 82, 84, 84, 87, 87, 88, 88, 91, 91, 93, 93, 94, 94,
 97, 97, 98, 98,100,100,103,103,104,104,107,107,109,109,110,110,
112,112,115,115,117,117,118,118,121,121,122,122,124,124,127,127,
128,128,131,131,133,133,134,134,137,137,138,138,140,140,143,143,
145,145,146,146,148,148,151,151,152,152,155,155,157,157,158,158,
161,161,162,162,164,164,167,167,168,168,171,171,173,173,174,174,
176,176,179,179,181,181,182,182,185,185,186,186,188,188,191,191,
193,193,194,194,196,196,199,199,200,200,203,203,205,205,206,206,
208,208,211,211,213,213,214,214,217,217,218,218,220,220,223,223,
224,224,227,227,229,229,230,230,233,233,234,234,236,236,239,239,
241,241,242,242,244,244,247,247,248,248,251,251,253,253,254,254};

void des_set_odd_parity(des_cblock *key)
{
  int i;

  for (i=0; i<sizeof(des_cblock); i++)
    (*key)[i]=odd_parity[(*key)[i]];
}

unsigned long des_cbc_cksum(des_cblock *input, des_cblock *output,
                            long length, DESContext *schedule,
                            des_cblock *ivec)
{
  register unsigned long tout0,tout1,tin0,tin1;
  register long l=length;
  word32   tin[2], tout[2];
  unsigned char *in,*out,*iv;

  in=(unsigned char *)input;
  out=(unsigned char *)output;
  iv=(unsigned char *)ivec;

  c2l(iv,tout0);
  c2l(iv,tout1);
  for (; l>0; l-=8)
    {
      if (l >= 8)
        {
          c2l(in,tin0);
          c2l(in,tin1);
        }
      else
        c2ln(in,tin0,tin1,l);

      tin0^=tout0; tin[0]=tin0;
```

```
      tin1^=tout1; tin[1]=tin1;
      des_encrypt(tin[0], tin[1], tout, schedule, 1);
      /* fix 15/10/91 eay - thanks to keithr@sco.COM */
      tout0=tout[0];
      tout1=tout[1];
    }
  if (out != NULL)
    {
      l2c(tout0,out);
      l2c(tout1,out);
    }
  tout0=tin0=tin1=tin[0]=tin[1]=0;
  return(tout1);
}
```

File Name: reporting.h

```
/* this file provides defines for error and debugging reporting */
#ifndef EXTERN
#define EXTERN extern
#endif EXTERN

#include "errormsg.h"
#include "debug.h"

#define ERROR(format, args...) fatal(LOG_ERR, format , ## args)
#define DEBUG(format, args...) /*DEBUG1(format , ## args)*/
```

File Name: vars.c

```
/*************************************************************************
 * LPRng - An Extended Print Spooler System
 *
 * Copyright 1988-1997, Patrick Powell, San Diego, CA
 *     papowell@sdsu.edu
 * See LICENSE for conditions of use.
 *
 *************************************************************************
 * MODULE: vars.c
 * PURPOSE: variables
 *************************************************************************/

static char *const _id =
"$Id: vars.c,v 3.7 1997/02/17 02:31:27 papowell Exp papowell $";

/* force local definitions */
#define EXTERN
#define DEFINE

#include "lp.h"
#include "printcap.h"
#include "permission.h"
#include "setuid.h"
#include "pr_support.h"
#include "timeout.h"
#include "fileopen.h"
#include "waitchild.h"
/**** ENDINCLUDE ****/

/*************************************************************************

Commentary:
Patrick Powell Tue Nov 26 08:10:12 PST 1996
Put all of the variables in a separate file.

 *************************************************************************/

/*
 * printcap variables used by LPD for printing
 * THESE MUST BE IN SORTED ORDER
 *  NOTE:  the maxval field is used to suppress clearing
 *   these values when initializing the printcap variable
 *   values.
 */

struct keywords Pc_var_list[] = {
{ "ab",  FLAG_K,  &Always_banner },
   /*  always print banner, ignore lpr -h option */
{ "ac",  STRING_K,  &Allow_class },
   /*  allow these classes to be printed */
{ "achk",  FLAG_K,  &Accounting_check },
   /*  query accounting server when connected */
{ "ae",  STRING_K,  &Accounting_end },
   /*  accounting at end (see also af, la, ar, as) */
{ "af",  STRING_K,  &Accounting_file },
   /*  name of accounting file (see also la, ar) */
```

```
{ "ah",  FLAG_K,  &Auto_hold },
   /*  automatically hold all jobs */
{ "allow_getenv", FLAG_K, &Allow_getenv, 1 },
{ "ar",  FLAG_K,  &Accounting_remote },
   /*  write remote transfer accounting (if af is set) */
{ "architecture", STRING_K, &Architecture },
{ "as",  STRING_K,  &Accounting_start },
   /*  accounting at start (see also af, la, ar) */
{ "be",  STRING_K,  &Banner_end },
   /*  end banner printing program overides bp */
{ "bk",  FLAG_K,  &Backwards_compatible },
   /*  backwards-compatible: job file strictly RFC-compliant */
{ "bk_filter_options", STRING_K, &BK_filter_options },
{ "bk_of_filter_options", STRING_K, &BK_of_filter_options },
{ "bkf",  FLAG_K,  &Backwards_compatible_filter },
   /*  backwards-compatible filters: use simple paramters */
{ "bl",  STRING_K,  &Banner_line },
   /*  short banner line sent to banner printer */
{ "bp",  STRING_K,  &Banner_printer },
   /*  banner printing program (see ep) */
{ "bq",  STRING_K,  &Bounce_queue_dest },
   /*  use filters on bounce queue files */
{ "br",  INTEGER_K,  &Baud_rate },
   /*  if lp is a tty, set the baud rate (see ty) */
{ "bs",  STRING_K,  &Banner_start },
   /*  banner printing program overrides bp */
{ "cd",  STRING_K,  &Control_dir },
   /*  control directory */
{ "check_for_nonprintable", FLAG_K, &Check_for_nonprintable },
{ "client_config_file", STRING_K, &Client_config_file, 1 },
{ "cm",  STRING_K,  &Comment_tag },
   /*  comment identifying printer (LPQ) */
{ "co",  INTEGER_K,  &Cost_factor },
   /*  cost in dollars per thousand pages */
{ "connect_failure_action", STRING_K, &Connect_failure_action },
   /* connection failure causes this script to be invoked */
{ "connect_grace", INTEGER_K, &Connect_grace },
   /* connection control for remote printers */
{ "connect_interval", INTEGER_K, &Connect_interval },
   /* connection control for remote printers */
{ "connect_retry", INTEGER_K, &Connect_try },
   /* connection control for remote printers */
{ "connect_timeout", INTEGER_K, &Connect_timeout },
   /* connection control for remote printers */
{ "control_filter", STRING_K, &Control_filter },
   /* control file filter */
{ "db",  STRING_K,  &New_debug },
   /*  debug level set for queue handler */
{ "default_auth", STRING_K, &Default_auth },
   /*  default authentication */
{ "default_banner_printer", STRING_K, &Default_banner_printer },
{ "default_format", STRING_K, &Default_format },
{ "default_logger_port", STRING_K, &Default_logger_port },
{ "default_logger_protocol", STRING_K, &Default_logger_protocol },
{ "default_permission", STRING_K, &Default_permission },
{ "default_printer", STRING_K, &Default_printer },
{ "default_priority", STRING_K, &Default_priority },
{ "default_remote_host", STRING_K, &Default_remote_host },
{ "default_tmp_dir", STRING_K, &Default_tmp_dir },
```

```
{ "direct_read", FLAG_K, &Direct_read },
{ "ek", FLAG_K, &Mini_Supervisor_Encrypt },
  /* Should the mini-supervisor encrypt? Only used with the ms flag */
{ "fc",  INTEGER_K,  &Clear_flag_bits },
   /*  if lp is a tty, clear flag bits (see ty) */
{ "fd",  FLAG_K,  &Forwarding_off },
   /*  if true, no forwarded jobs accepted */
{ "ff",  STRING_K,  &Form_feed },
   /*  string to send for a form feed */
{ "filter_control", STRING_K, &Filter_control },
{ "filter_ld_path", STRING_K, &Filter_ld_path, 1 },
{ "filter_options", STRING_K, &Filter_options },
{ "filter_path", STRING_K, &Filter_path, 1 },
{ "fo",  FLAG_K,  &FF_on_open },
   /*  print a form feed when device is opened */
{ "force_queuename", STRING_K, &Force_queuename },
   /*  force use of this queuename if none provided */
{ "forward_auth", STRING_K, &Forward_auth },
   /*  do server to server authentication if authenticted by user */
{ "fq",  FLAG_K,  &FF_on_close },
   /*  print a form feed when device is closed */
{ "fs",  INTEGER_K,  &Set_flag_bits },
   /*  like `fc' but set bits (see ty) */
{ "fx",  STRING_K,  &Formats_allowed },
   /*  valid output filter formats */
{ "generate_banner", FLAG_K, &Generate_banner },
{ "group", STRING_K, &Daemon_group, 1 },
{ "hl",  FLAG_K,  &Banner_last },
   /*  print banner after job instead of before */
{ "if",  STRING_K,  &IF_Filter },
   /*  filter command, run on a per-file basis */
{ "ipv6",  FLAG_K,  &IPV6Protocol },
   /*  filter command, run on a per-file basis */
{ "kerberos_keytab", STRING_K, &Kerberos_keytab },
        /* keytab file location for kerberos, used by server */
{ "kerberos_life", STRING_K, &Kerberos_life },
        /* key lifetime for kerberos, used by server */
{ "kerberos_renew", STRING_K, &Kerberos_renew },
        /* key renewal time for kerberos, used by server */
{ "kerberos_server_principle", STRING_K, &Kerberos_server_principle },
        /* remote server principle, overides default */
{ "kerberos_service", STRING_K, &Kerberos_service },
        /* default service */
{ "la",  FLAG_K,  &Local_accounting },
   /*  write local printer accounting (if af is set) */
{ "ld",  STRING_K,  &Leader_on_open },
   /*  leader string printed on printer open */
{ "lf",  STRING_K,  &Log_file },
   /*  error log file (servers, filters and prefilters) */
{ "lk", FLAG_K,  &Lock_it },
   /* lock the IO device */
{ "localhost", STRING_K, &Localhost, 1 },
{ "lockfile", STRING_K, &Lockfile, 1 },
{ "logfile", STRING_K, &Logfile },
{ "logger_destination",  STRING_K,  &Logger_destination },
   /* where to send status information for logging */
{ "longnumber",  FLAG_K,  &Long_number },
   /*  use long job number when a job is submitted */
{ "lp",  STRING_K,  &Lp_device },
```

```
   /*  device name or lp-pipe command to send output to */
{ "lpd_port", STRING_K, &Lpd_port },
{ "lpd_printcap_path", STRING_K, &Lpd_printcap_path, 1 },
{ "lpr_bounce", FLAG_K, &Lpr_bounce },
{ "mail_operator_on_error", STRING_K, &Mail_operator_on_error },
{ "max_servers_active", INTEGER_K, &Max_servers_active },
{ "max_status_size", INTEGER_K, &Max_status_size },
{ "mc",  INTEGER_K,  &Max_copies },
   /*  maximum copies allowed */
{ "mi",  STRING_K,  &Minfree },
   /*  minimum space (Kb) to be left in spool filesystem */
{ "min_status_size", INTEGER_K, &Min_status_size },
{ "minfree", STRING_K, &Minfree },
   /**/
{ "ml",  INTEGER_K,  &Min_printable_count },
   /*  minimum printable characters for printable check */
{ "ms", FLAG_K, &Is_Mini_Supervisor },
  /* is the lp-pipe command referring to a mini-supervisor? */
{ "mx",  INTEGER_K,  &Max_job_size },
   /*  maximum job size (1Kb blocks, 0 = unlimited) */
{ "nb",  INTEGER_K,  &Nonblocking_open },
   /*  use nonblocking open */
{ "nw",  FLAG_K,  &NFS_spool_dir },
   /*  spool dir is on an NFS file system (see rm, rp) */
{ "of",  STRING_K,  &OF_Filter },
   /*  output filter, run once for all output */
{ "of_filter_options", STRING_K, &OF_filter_options },
{ "originate_port", STRING_K, &Originate_port },
{ "pass_env",  STRING_K,  &Pass_env },
   /* if client, pass these environment variables */
{ "pl",  INTEGER_K,  &Page_length },
   /*  page length (in lines) */
{ "poll_time",  INTEGER_K,  &Poll_time },
   /*  interval in secs between starting up all servers */
{ "pr",  STRING_K,  &Pr_program },
   /*  pr program for p format */
{ "printcap_path", STRING_K, &Printcap_path, 1 },
{ "printer_perms_path", STRING_K, &Printer_perms_path, 1 },
{ "ps",  STRING_K,  &Status_file },
   /*  printer status file name */
{ "pw",  INTEGER_K,  &Page_width },
   /*  page width (in characters) */
{ "px",  INTEGER_K,  &Page_x },
   /*  page width in pixels (horizontal) */
{ "py",  INTEGER_K,  &Page_y },
   /*  page length in pixels (vertical) */
{ "qq",  FLAG_K,  &Use_queuename },
   /*  put queue name in control file */
{ "remote_user",  STRING_K,  &Remote_user },
   /*  remote-queue machine (hostname) (with rm) */
{ "rm",  STRING_K,  &RemoteHost },
   /*  remote-queue machine (hostname) (with rm) */
{ "router",  STRING_K,  &Routing_filter },
   /*  routing filter, returns destinations */
{ "rp",  STRING_K,  &RemotePrinter },
   /*  remote-queue printer name (with rp) */
{ "rt",  INTEGER_K,  &Send_try },
   /*  number of times to try printing or transfer (0=infinite) */
{ "rw",  FLAG_K,  &Read_write },
```

```
   /*  open the printer for reading and writing */
{ "save_on_error",  FLAG_K,  &Save_on_error },
   /*  save job when an error */
{ "save_when_done",  FLAG_K,  &Save_when_done },
   /*  save job when done */
{ "sb",  FLAG_K,  &Short_banner },
   /*  short banner (one line only) */
{ "sc",  FLAG_K,  &Suppress_copies },
   /*  suppress multiple copies */
{ "sd",  STRING_K,  &Spool_dir },       /* EXPAND */
   /*  spool directory (only ONE printer per directory!) */
{ "send_block_format", FLAG_K, &Send_block_format },
   /* send block of data, rather than individual files */
{ "send_data_first", FLAG_K, &Send_data_first },
   /* failure action to take after send_try attempts failed */
{ "send_failure_action", STRING_K, &Send_failure_action },
   /* failure action to take after send_try attempts failed */
{ "send_timeout", INTEGER_K, &Send_timeout },
   /* timeout for each write to device to complete */
{ "send_try", INTEGER_K, &Send_try },
   /* numbers of times to try sending job - 0 is infinite */
{ "sendmail", STRING_K, &Sendmail, 1 },
{ "server_auth_command", STRING_K, &Server_authentication_command, 1 },
   /*  authenticate transfer command */
{ "server_config_file", STRING_K, &Server_config_file, 1 },
{ "server_tmp_dir", STRING_K, &Server_tmp_dir, 1 },
{ "server_user", STRING_K, &Server_user },
   /* server user for authentication */
{ "sf",  FLAG_K,  &No_FF_separator },
   /*  suppress form feeds separating multiple jobs */
{ "sh",  FLAG_K,  &Suppress_header },
   /*  suppress headers and/or banner page */
{ "spool_dir_perms", INTEGER_K, &Spool_dir_perms, 1 },
{ "spool_file_perms", INTEGER_K, &Spool_file_perms, 1 },
{ "spread_jobs", INTEGER_K, &Spread_jobs },
{ "ss",  STRING_K,  &Server_queue_name },
   /*  name of queue that server serves (with sv) */
{ "sv",  STRING_K,  &Server_names },
   /*  names of servers for queue (with ss) */
{ "sy",  STRING_K,  &Stty_command },
   /*  stty commands to set output line characteristics */
{ "syslog_device", STRING_K, &Syslog_device, 1 },
{ "tr",  STRING_K,  &Trailer_on_close },
   /*  trailer string to print when queue empties */
{ "translate_format",  STRING_K,  &Xlate_format },
   /*  translate format from one to another - similar to tr(1) utility */
{ "ty",  STRING_K,  &Stty_command },
   /*  stty commands to set output line characteristics */
{ "use_auth", STRING_K, &Use_auth },
   /*  use authentication */
{ "use_date",  FLAG_K,  &Use_date },
   /*  put date in control file */
{ "use_identifier",  FLAG_K,  &Use_identifier },
   /*  put identifier in control file */
{ "use_info_cache", FLAG_K, &Use_info_cache },
   /*  read and cache information */
{ "use_queuename", FLAG_K, &Use_queuename },
   /*  put queue name in control file */
{ "use_shorthost",  FLAG_K,  &Use_shorthost },
```

```
   /*  Use short hostname for lpr control and data file names */
{ "user", STRING_K, &Daemon_user, 1 },
   /*  server user for SUID purposes */
{ "user_auth_command", STRING_K, &User_authentication_command },
   /*  authenticate transfer command */
{ "xc",  INTEGER_K,  &Clear_local_bits },
   /*  if lp is a tty, clear local mode bits (see ty) */
{ "xs",  INTEGER_K,  &Set_local_bits },
   /*  like `xc' but set bits (see ty) */
{ "xt",  FLAG_K,  &Check_for_nonprintable },
   /*  formats to check for printable files */
{ "xu",  STRING_K,  &Local_permission_file },
   /*  additional permissions file for this queue */
{ (char *)0 }
} ;


struct keywords Lpd_parms[] = {
{ "Clean",  INTEGER_K , &Clean },
{ "Foreground",  INTEGER_K , &Foreground },
{ "FQDNHost",  STRING_K , &FQDNHost },
{ "FQDNRemote",  STRING_K , &FQDNRemote },
{ "Logname",  STRING_K , &Logname },
{ "Printer",  STRING_K , &Printer },
{ "ShortHost",  STRING_K , &ShortHost },
{ "ShortRemote",  STRING_K , &ShortRemote },
{ 0 }
} ;

/* force reference to Copyright */
char **Copyright_ref = Copyright;

struct keywords Keyletter[] = {
        { "P", STRING_K, &Printer },
        { "h", STRING_K, &ShortHost },
        { "H", STRING_K, &FQDNHost },
        { "a", STRING_K, &Architecture },
        { "R", STRING_K, &RemotePrinter },
        { "M", STRING_K, &RemoteHost },
        { 0 }
};
```

Appendix B:
Source Code Specific to the Mini-supervisor

File Name: encrypted.c

```
#include "mini.h"

void ServiceEncrypted(int serverFD, int printerFD)
{
  char          buffer[MAXLEN];
  ssize_t       bytesWritten, bytesRead;
  int           gotEOF = 0;
  char          jobDataBuffer[MAX_JOB_LEN];
  char          responseDataBuffer[MAX_RESPONSE_LEN];
  long          jobDataBytes;
  long          responseDataBytes;
  Keys          jobKeys;
  Keys          responseKeys;
  int           theError;
  int           jobBufferFull = 0;

  /* Send a send key message */
  sprintf(buffer, "%c%d\n", SEND_KEY_MSG, GetJobNumber());
  bytesWritten = WriteMessage(serverFD, buffer, strlen(buffer));
  if(bytesWritten != strlen(buffer))
    {
      ERROR("ServiceEncrypted: failed writing send key message.");
      return;
    }

  DEBUG("ServiceEncrypted: sent send key message.");

  /* Try to get a key message */
  bytesRead = ReadMessage(serverFD, buffer);
  if(bytesRead <= 0)
    {
      ERROR("ServiceEncrypted: failed reading key message.");
      return;
    }

  /* Check to see what we got */
  switch(buffer[0])
    {
    case FATAL_ERROR_MSG:
      ERROR("ServiceEncrypted: got fatal error message. Aborting.");
      return;

    case KEY_MSG:
      DEBUG("ServiceEncrypted: got key message.");
      break;

    default:
     ERROR("ServiceEncrypted: Invalid protocol message.  Sending fatal error.");
      SendFatalError(serverFD);
      return;
    }
```

```
theError = GetKeys(serverFD, &jobKeys, &responseKeys);

if(theError != NO_ERR)
  {
    /* We couldn't get the key */
    SendFatalError(serverFD);
    return;
  }

sprintf(buffer, "%c\n", BEGIN_JOB_MSG);
bytesWritten = WriteMessage(serverFD, buffer, strlen(buffer));

if(bytesWritten <= 0)
  {
    ERROR("ServiceEncrypted: failed writing being job message.");
    return;
  }

jobDataBytes = 0;
responseDataBytes = 0;
jobBufferFull = 0;

while(!(gotEOF && jobDataBytes == 0 && responseDataBytes == 0))
  {
    fd_set readfds, writefds;
    int readyConnections;

    FD_ZERO(&readfds);
    FD_ZERO(&writefds);

    /* check for data to read if we have space */
    if(jobDataBytes < MAX_JOB_LEN && !gotEOF)
      FD_SET(serverFD, &readfds);
    /*if(responseDataBytes < MAX_RESPONSE_LEN)
      FD_SET(printerFD, &readfds);*/

    /* If we have data to write, check for that */
    if(jobDataBytes > 0)
      FD_SET(printerFD, &writefds);
    if(responseDataBytes > 0)
      FD_SET(serverFD, &writefds);

    readyConnections = select(maximum(printerFD, serverFD) + 1,
                              &readfds, &writefds, 0, 0);
    theError = errno;
    DEBUG("select() returned [%d]: %s",
          readyConnections,
          (readyConnections < 0) ? ErrorToText(theError) : "No error");

    if(readyConnections < 0)
      {
        if(theError != EINTR)
          {
            ERROR("ServiceEncrypted: select() failed: %s",
                  ErrorToText(theError));
            exit(1);
          }
        continue;   /* try again */
```

```
  }

if(readyConnections == 0)
  continue;

/* can we read a data messsage from the server? */
if(jobDataBytes < MAX_JOB_LEN &&
   !jobBufferFull &&
   FD_ISSET(serverFD, &readfds))
  {
    DEBUG("ServiceEncrypted: reading data from the server.");

    theError = EGetData(serverFD,
                        jobDataBuffer,
                        &jobDataBytes,
                        MAX_JOB_LEN,
                        &jobKeys);

    if(theError == GOT_EOF)
      gotEOF = 1;

    if(theError == FATAL)
      exit(1);

    if(theError == FULL)
      jobBufferFull = 1;

    continue;
  }

/* can we read data from the printer? */
if(responseDataBytes < MAX_RESPONSE_LEN && FD_ISSET(printerFD, &readfds))
  {
    DEBUG("ServiceEncrypted: reading data from the printer.");

    theError = GetData(printerFD,
                       responseDataBuffer,
                       &responseDataBytes,
                       MAX_RESPONSE_LEN);

    if(theError == FATAL)
      exit(1);
  }

/* can we write data to the printer? */
if(jobDataBytes > 0 && FD_ISSET(printerFD, &writefds))
  {
    DEBUG("ServiceEncrypted: writing data to the printer.");

    theError = SendData(printerFD,
                        jobDataBuffer,
                        &jobDataBytes,
                        MAX_JOB_LEN);
    if(theError == FATAL)
      exit(1);

    jobBufferFull = 0;
  }
```

```
      /* can we write a response message to the server? */
      if(responseDataBytes > 0 && FD_ISSET(serverFD, &writefds))
        {
          DEBUG("ServiceEncrypted: writing data to the server.");

          theError = ESendData(serverFD,
                               responseDataBuffer,
                               &responseDataBytes,
                               MAX_RESPONSE_LEN,
                               &responseKeys);

          if(theError != NO_ERR)
            exit(1);
        }

    }

  return;
}


int GetKeys(int serverFD, Keys *jobKeys, Keys *responseKeys)
{
  long          currentPtr;
  char          keyBuffer[MAX_KEY_LEN];
  Keys          masterKeys;
  des_cblock    jk1, jk2, jk3, rk1, rk2, rk3;
  FourBytes     serverJobNumber, miniJobNumber;
  int           theError;
  unsigned long keyDataSize;

  theError = GetMasterKeys(&masterKeys);
  if(theError != NO_ERR)
    return(FATAL);

  /* get the key data and decrypt it */
  keyDataSize = 0;  /* the buffer is currently empty */
  theError = EGetData(serverFD, keyBuffer, &keyDataSize,
                      MAX_KEY_LEN, &masterKeys);

  ZeroKeys(masterKeys);

  if(theError != NO_ERR)
    return(FATAL);

  /* put the keys and job number in the buffer */
  currentPtr = 0;
  memcpy(jk1, &keyBuffer[currentPtr], sizeof(des_cblock));
  currentPtr += sizeof(des_cblock);
  memcpy(jk2, &keyBuffer[currentPtr], sizeof(des_cblock));
  currentPtr += sizeof(des_cblock);
  memcpy(jk3, &keyBuffer[currentPtr], sizeof(des_cblock));
  currentPtr += sizeof(des_cblock);
  memcpy(rk1, &keyBuffer[currentPtr], sizeof(des_cblock));
  currentPtr += sizeof(des_cblock);
  memcpy(rk2, &keyBuffer[currentPtr], sizeof(des_cblock));
  currentPtr += sizeof(des_cblock);
  memcpy(rk3, &keyBuffer[currentPtr], sizeof(des_cblock));
  currentPtr += sizeof(des_cblock);
```

```
  memcpy(&serverJobNumber, &keyBuffer[currentPtr], sizeof(FourBytes));
  serverJobNumber = ntohl(serverJobNumber);
  currentPtr += sizeof(FourBytes);

  if(currentPtr != keyDataSize)
    {
      ERROR("GetKeys: Key data size mismatch.");
      return(FATAL);
    }

  /* zero out the keyBuffer so the memory is safe */
  memset(keyBuffer, 0, MAX_KEY_LEN);

  /* compare job Numbers */
  miniJobNumber = GetJobNumber();
  DEBUG("GetKeys: serverJobNumber is %d; miniJobNumber is %d.",
        serverJobNumber, miniJobNumber);

  if(miniJobNumber != serverJobNumber)
    {
      char buffer[MAXLEN];
      ssize_t bytesWritten;

      ERROR("GetKeys: Job number mismatch.");
      sprintf(buffer, "%c\n", JOB_NUMBER_MISMATCH_MSG);
      bytesWritten = WriteMessage(serverFD, buffer, strlen(buffer));
      if(bytesWritten <= 0)
        {
          ERROR("GetKeys: sending job number mismatch failed.");
          return(FATAL);
        }

      return(FATAL);
    }

  /* Now generate the key schedules */
  des_set_key(jk1, &jobKeys->key1);
  des_set_key(jk2, &jobKeys->key2);
  des_set_key(jk3, &jobKeys->key3);
  des_set_key(rk1, &responseKeys->key1);
  des_set_key(rk2, &responseKeys->key2);
  des_set_key(rk3, &responseKeys->key3);

  /* initalize the vectors */
  memset(jobKeys->ivec1, 0, sizeof(des_cblock));
  memset(jobKeys->ivec2, 0, sizeof(des_cblock));
  memset(jobKeys->ivec3, 0, sizeof(des_cblock));
  memset(responseKeys->ivec1, 0, sizeof(des_cblock));
  memset(responseKeys->ivec2, 0, sizeof(des_cblock));
  memset(responseKeys->ivec3, 0, sizeof(des_cblock));

  return(NO_ERR);
}
```

File Name: errors.c

```
/* errors.c -- Translates errors to text messages */

#include "mini.h"

char *ErrorToText(int theError)
{
  char *errorText;

  /* currently just this, but we have have a case statement later for
     minilpd specific calls */
  errorText = strerror(theError);

  return(errorText);
}
```

File Name: jobnumber.c

```
#include "mini.h"


FourBytes GetJobNumber(void)
{
  FourBytes jobNum;

  jobNum = gJobNumber;

  return(jobNum);
}


void SetJobNumber(FourBytes jobNum)
{
  gJobNumber = jobNum;

  return;
}
```

File Name: Makefile

```
MINISRCS = encrypted.c jobnumber.c mini.c signals.c\
           unencrypted.c errors.c messages.c data.c utils.o\
           des.c


MINIOBJS = encrypted.o jobnumber.o mini.o signals.o\
           unencrypted.o errors.o messages.o data.o utils.o\
           des.o

CC = gcc -g -Wall
LD = gcc -g

CFLAGS = -I../include/ -I/usr/local/include
#LDFLAGS = -lnsl -lsocket

all:: mini

mini:: ${MINIOBJS}
        ${LD} -o mini ${MINIOBJS} ${LDFLAGS}

clean::
        rm -f mini a.out core ${MINIOBJS}
```

File Name: mini.c

```
/* mini.c - mini-supervisor main request loop. */

#include "mini.h"

/* Eww, a global variable!  But who wants to pass this around? */
pid_t gServiceChildPID = NOCHILD;              /* The child's PID */
int   gProcessType = PARENT;                   /* The type of process */
int   gNoUnencryptedConnections = 0;           /* Allow unencrypted? */
unsigned long gJobNumber = 0;                  /* The Job Number */

void main(int argc, char *argv[], char *envp[])
{
  int listenFD = 0, connectFD = 0;     /* Socket to listen/connect on */
  struct sockaddr_in sin;
  fd_set defreadfds, readfds;          /* For use with select(); */
  int noFork = 0;                      /* To fork or not to fork? */
  int theError = 666, one = 1;
  struct sigaction act;
  sigset_t sigChildSet;

  while (--argc > 0)
    {
      argv++;
      if (argv[0][0] == '-')
        switch (argv[0][1])
          {
          case 'F':
          case 'f':
            noFork = 1;
            break;

          case 'E':
          case 'e':
            gNoUnencryptedConnections = 1;
            break;

          default:
            break;
          }
    }

  /* Detach from the parent if not nofork */
  if(!noFork)
    if(fork())
      exit(0);

  /* set up the signal handler for SIGCHILD */
  /* we need to know when the child exits */
  sigemptyset(&act.sa_mask);
  act.sa_flags = 0;
  act.sa_handler = SigChildHandler;
  sigaction(SIGCHLD, &act, 0);

  /* for blocking and unblocking SIGCHLD later */
  sigemptyset(&sigChildSet);
  sigaddset(&sigChildSet, SIGCHLD);
```

```
/* set up the socket to listen on */
listenFD = socket(AF_INET, SOCK_STREAM, 0);
if(listenFD < 0)
  {
    theError = errno;
    ERROR("Socket() failed: %s", ErrorToText(theError));
    exit(1);
  }

/* Bind to the port we want to listen on */
setsockopt(listenFD, SOL_SOCKET, SO_REUSEADDR, (char *) &one, sizeof(one));

sin.sin_family = AF_INET;
sin.sin_addr.s_addr = INADDR_ANY;
sin.sin_port = htons(MINIPORT);
if(bind(listenFD, (struct sockaddr *)&sin, sizeof(sin)) < 0)
  {
    theError = errno;
    ERROR("Bind() failed: %s", ErrorToText(theError));
    exit(1);
  }

/* Listen to that port */
FD_ZERO(&defreadfds);
FD_SET(listenFD, &defreadfds);
if(listen(listenFD, 5) != 0)
  {
    theError = errno;
    ERROR("listen() failed: %s", ErrorToText(theError));
    exit(1);
  }

/* syslog that minilpd has started */
DEBUG("Mini-supervisor started.");

/* The main loop: where we accept connections, etc */
for(;;)
  {
    int connection;
    struct timeval timeOut;

    readfds = defreadfds;
    timeOut.tv_sec = 5;
    timeOut.tv_usec = 0;

    /* Do any idle activity here */

    /* check for incomming connections */
    connection = select(listenFD + 1, &readfds, 0, 0, &timeOut);
    theError = errno;

    /*      DEBUG("select() returned [%d]: %s",
           connection,
           (connection < 0) ? ErrorToText(theError) : "No error"); */

    /* Check for problems! */
    if(connection < 0)
      {
```

```
    if(theError != EINTR)
      {
        ERROR("select() failed: %s", ErrorToText(theError));
        exit(1);
      }
    continue;   /* try again */
  }

/* timeout? */
if(connection == 0)
    continue;   /* try again */

/* was it the listen socket? (should be) */
if(FD_ISSET(listenFD, &readfds))
  {
    int socketSize = sizeof(sin);

    /* Accept the connection */
    DEBUG("Accepting connection on socket %d", listenFD);
    connectFD = accept(listenFD, (struct sockaddr *)&sin, &socketSize);

    /* did we fail? */
    if(connectFD < 0)
      {
        theError = errno;
        ERROR("accept() failed: %s", ErrorToText(theError));
        continue;
      }

    /* Are we already handling a connection? */
    if(gServiceChildPID == NOCHILD)
      {
        FourBytes jobNum;

        /* Assign this job a new unique job number */
        jobNum = GetJobNumber();
        if(jobNum >= MAXJOBNUMBER)
          jobNum = 0;  /* reset if we exceed the max length */
        else
          jobNum++;
        SetJobNumber(jobNum);

        /* Block out SIGCHLD while we fork or we might lose */
        sigprocmask(SIG_BLOCK, &sigChildSet, 0);

        /* Fork a child to handle the connection */
        DEBUG("Forking a child to handle the connection.");
        gServiceChildPID = fork();

        /* Let's get SIGCHLDs back now that we are safe */
        sigprocmask(SIG_UNBLOCK, &sigChildSet, 0);

       /* did something horrible happen? (like out of memory/processes) */
        if(gServiceChildPID < 0)
          {
            ERROR("fork() failed trying to service a connection.");
            gServiceChildPID = NOCHILD;
            continue;
          }
```

```
            if(gServiceChildPID == 0)
              {
                /* We are the service child */
                gProcessType = SERVICE_CHILD;

                /* Service the connection and then exit */
                ServiceConnection(connectFD);
                close(connectFD);

                exit(0);
              }

            /* Else, we are the parent, keep looping */
          }
        else
          {
            /* Fork a child to tell the guy to retry later */
            DEBUG("Child %d is handling a connection. Forking to send Busy",
                  (int)gServiceChildPID);

            if(!fork())
              {
                /* we are a busy-message child */
                gProcessType = BUSY_CHILD;

                SendBusy(connectFD);
                close(connectFD);

                exit(0);
              }
          }
      }
  }
  exit(0);
}


void SendBusy(int connectFD)
{
  ssize_t bytesRead, bytesWritten;
  char buffer[MAXLEN];

  /* Make sure we are a busy message child */
  if(gProcessType != BUSY_CHILD)
    {
      ERROR("Process should not be sending busy messages.");
      return;
    }

  bytesRead = ReadMessage(connectFD, buffer);
  if(bytesRead <= 0)
    {
      ERROR("SendBusy(): reading request failed.");
      return;
    }

  /* Check to see if we are getting a start job request */
  if(buffer[0] == START_ENCRYPTED_JOB_MSG ||
```

```
     buffer[0] == START_UNENCRYPTED_JOB_MSG)
     {
      /* This was a job request */
      DEBUG("SendBusy(): received job request: %s", buffer);

      /* Send a busy message */
      sprintf(buffer, "%c\n", BUSY_MSG);
      bytesWritten = WriteMessage(connectFD, buffer, strlen(buffer));
      if(bytesWritten < strlen(buffer))
        return;

      DEBUG("SendBusy(): sent busy message.");
      return;
    }
  else
    {
      /* This was an illegal request */
      DEBUG("SendBusy(): received illegal request: %s", buffer);

      /* Send a fatal error message */
      SendFatalError(connectFD);

      return;
    }

  ERROR("SendBusy(): never sent reply.");
  return;
}


void ServiceConnection(int connectFD)
{
  ssize_t bytesRead;
  char buffer[MAXLEN];
  int printerFD, i;

  /* Sanity Check to make sure we are a service connection child */
  if(gProcessType != SERVICE_CHILD)
    {
      ERROR( "Process should not be servicing connections.");
      exit(1);
    }

  bytesRead = ReadMessage(connectFD, buffer);
  if(bytesRead <= 0)
    {
      ERROR( "ServiceConnection(): failed reading start job message.");
      return;
    }

  if(buffer[0] != START_ENCRYPTED_JOB_MSG &&
     buffer[0] != START_UNENCRYPTED_JOB_MSG)
    {
      /* This was an illegal request */
      /* Send a fatal error message if we didn't get a fatal error message */
      if(buffer[0] != FATAL_ERROR_MSG)
        {
          DEBUG(
                 "ServiceConnection(): received illegal request: %s", buffer);
```

```
        SendFatalError(connectFD);
      }
    else
      DEBUG(
           "ServiceConnection(): received fatal error, aborting.\n");
    return;
  }

/* this is a job request message */
/* open the parallel port so we can write to the printer */
for(i = 1; ; i = i < 32 ? i << 1 : i)
  {
    printerFD = open(PARALLEL, O_WRONLY, 0);

    /* did we succeed? */
    if(printerFD >= 0)
      break;

    /* is the printer dead? */
    if(errno == ENOENT)
      {
        ERROR( "ServiceConnection(): printer not responding.");
        SendFatalError(connectFD);
        return;
      }

    if(i == 1)
      DEBUG("ServiceConnection(): printer is sleeping.  Waiting.");

    sleep(i);
  }

DEBUG("Opened a connection to the printer");

/* Check to see if we are getting a start encrypted job request */
if(buffer[0] == START_ENCRYPTED_JOB_MSG)
  {
    DEBUG("ServiceConnection(): received encrypted job request: %s",
          buffer);

    /* Handle the encrypted connection */
    ServiceEncrypted(connectFD, printerFD);
  }
else
  /* Check to see if we are getting a start unencrypted job request */
  if(buffer[0] == START_UNENCRYPTED_JOB_MSG)
    {
      DEBUG("ServiceConnection(): received unencrypted job request: %s",
            buffer);

      /* Handle the unencrypted connection */
      ServiceUnencrypted(connectFD, printerFD);
    }

/* give the printer a ^D to tell it the job is over */
sprintf(buffer, "%c", '\004');
if(write(printerFD, buffer, 1) != 1)
  ERROR("ServiceConnection(): failed to send printer termination character.");
else
```

```
    DEBUG("ServiceConnection(): sent printer termination character.");

  /* close the connection with the printer */
  close(printerFD);

  return;
}
```

File Name: mini.h

```
/* minilpd.h -- Global includes and definitions for the mini-supervisor */

#include "shared.h"


/* DEFINES */
#define NOCHILD      -1                  /* not a legal pid, so good */
#define PARENT        1
#define SERVICE_CHILD 2
#define BUSY_CHILD    3
#define PARALLEL      "/dev/lpt0"
#define MAXJOBNUMBER  1000000000


/* GLOBAL VARIABLES */
extern pid_t gServiceChildPID;              /* The child's PID */
extern int gProcessType;                    /* The type of process */
int    gNoUnencryptedConnections;           /* Allow unencrypted? */
extern unsigned long gJobNumber;            /* The Job Number */

/* FUNCTION PROTOTYPES */

/* minilpd.c */
void main(int argc, char *argv[], char *envp[]);
void SendBusy(int connectFD);
void ServiceConnection(int connectFD);

/* encrypted.c */
void ServiceEncrypted(int serverFD, int printerFD);
int GetKeys(int serverFD, Keys *jobKeys, Keys *responseKeys);

/* unencrypted.c */
void ServiceUnencrypted(int connectFD, int printerFD);

/* signals.c */
void SigChildHandler(int signalNumber);

/* errors.c */
char *ErrorToText(int theError);

/* jobnumber.c */
FourBytes GetJobNumber(void);
void SetJobNumber(FourBytes jobNum);
```

File Name: reporting.h

```
/* this file provides defines for error and debugging reporting */

#define ERROR(format, args...) printf(format "\n" , ## args)
#define DEBUG(format, args...) /*printf(format "\n" , ## args)*/
```

File Name: signals.c

```
#include "mini.h"


void SigChildHandler(int signalNumber)
{
  int status;
  pid_t pid;

  /* Sanity check to make sure there are no grandchildren */
  if(gProcessType != PARENT)
    {
      ERROR("SigChildHandler called for child.  gServiceChildPID: %d.",
            (int) gServiceChildPID);
      exit(0);
    }

  /* Sanity Check to make sure the handler is properly installed */
  if(signalNumber != SIGCHLD)
    {
      ERROR("SigChildHandler called with signal: %d.", signalNumber);
      exit(0);
    }

  /* What happened to the child? */
  pid = wait(&status);

  if(pid == gServiceChildPID)  /* Is it the service child exiting? */
    {
      if(WIFEXITED(status) || WIFSIGNALED(status))
        {
          DEBUG("Service Child %d exited.", (int) pid);
          gServiceChildPID = NOCHILD;       /* The child has died */
        }
    }

  return;
}
```

File Name: unencrypted.c

```
#include "mini.h"

void ServiceUnencrypted(int serverFD, int printerFD)
{
  char    buffer[MAXLEN];
  ssize_t bytesWritten;
  int     gotEOF = 0;
  char    jobDataBuffer[MAX_JOB_LEN];
  char    responseDataBuffer[MAX_RESPONSE_LEN];
  long    jobDataBytes;
  long    responseDataBytes;
  int     theError;

  /* If there are no unencrypted connections allowed, refuse */
  if(gNoUnencryptedConnections)
    {
      /* Clear text printing is not allowed. */
      DEBUG("ServiceUnencrypted: unencrypted connections disabled: %s",
            buffer);

      /* Send a no encrypted jobs message */
      sprintf(buffer, "%c\n", NO_UNENCRYPTED_JOB_MSG);
      bytesWritten = WriteMessage(serverFD, buffer, strlen(buffer));
      if(bytesWritten != strlen(buffer))
        {
          ERROR("ServiceUnencrypted: failed writing no unencrypted job msg.");
          return;
        }

      DEBUG("ServiceUnencrypted: sent no encrypted job message.");
      return;
    }

  /* Everything is ready. */
  /* Send a begin job message */
  sprintf(buffer, "%c\n", BEGIN_JOB_MSG);
  bytesWritten = WriteMessage(serverFD, buffer, strlen(buffer));
  if(bytesWritten != strlen(buffer))
    {
      ERROR("ServiceUnencrypted: failed writing begin job message.");
      return;
    }

  DEBUG("ServiceUnencrypted: sent begin job message.");

  jobDataBytes = 0;
  responseDataBytes = 0;

  while(!(gotEOF && jobDataBytes == 0 && responseDataBytes == 0))
    {
      fd_set readfds, writefds;
      int readyConnections;

      FD_ZERO(&readfds);
      FD_ZERO(&writefds);
```

```
/* check for data to read if we have space */
if(jobDataBytes < MAX_JOB_LEN && !gotEOF)
  FD_SET(serverFD, &readfds);
/*if(responseDataBytes < MAX_RESPONSE_LEN)
  FD_SET(printerFD, &readfds);*/

/* If we have data to write, check for that */
if(jobDataBytes > 0)
  FD_SET(printerFD, &writefds);
if(responseDataBytes > 0)
  FD_SET(serverFD, &writefds);

readyConnections = select(maximum(printerFD, serverFD) + 1,
                          &readfds, &writefds, 0, 0);
theError = errno;
DEBUG("select() returned [%d]: %s",
       readyConnections,
       (readyConnections < 0) ? ErrorToText(theError) : "No error");

if(readyConnections < 0)
  {
    if(theError != EINTR)
      {
        ERROR("ServiceUnencrypted: select() failed: %s",
              ErrorToText(theError));
        exit(1);
      }
    continue;   /* try again */
  }

if(readyConnections == 0)
  continue;

/* can we read a data messsage from the server? */
if(jobDataBytes < MAX_JOB_LEN && FD_ISSET(serverFD, &readfds))
  {
    DEBUG("ServiceUnencrypted: reading data from the server.");

    theError = GetData(serverFD,
                       jobDataBuffer,
                       &jobDataBytes,
                       MAX_JOB_LEN);

    if(theError == GOT_EOF)
      gotEOF = 1;

    if(theError == FATAL)
      exit(1);

    continue;
  }

/* can we read data from the printer? */
if(responseDataBytes < MAX_RESPONSE_LEN && FD_ISSET(printerFD, &readfds))
  {
    DEBUG("ServiceUnencrypted: reading data from the printer.");

    theError = GetData(printerFD,
                       responseDataBuffer,
```

```
                        &responseDataBytes,
                        MAX_RESPONSE_LEN);

        if(theError == FATAL)
          exit(1);
      }

    /* can we write data to the printer? */
    if(jobDataBytes > 0 && FD_ISSET(printerFD, &writefds))
      {
        DEBUG("ServiceUnencrypted: writing data to the printer.");

        theError = SendData(printerFD,
                        jobDataBuffer,
                        &jobDataBytes,
                        MAX_JOB_LEN);
        if(theError == FATAL)
          exit(1);
      }

    /* can we write a response message to the server? */
    if(responseDataBytes > 0 && FD_ISSET(serverFD, &writefds))
      {
        DEBUG("ServiceUnencrypted: writing data to the server.");

        theError = SendData(serverFD,
                        responseDataBuffer,
                        &responseDataBytes,
                        MAX_RESPONSE_LEN);

        if(theError == FATAL)
          exit(1);
      }

  }
}
```

Appendix C:
Shared Source Code

File Name: data.c

```
#include "shared.h"

int GetData(int getFromFD, char *dataBuffer, long *dataBytes, long bufferBytes)
{
  ssize_t bytesRead;
  int succeeded = 0;

  /* Sanity checks */
  if(*dataBytes >= bufferBytes)
    {
      ERROR("GetData: buffer is already full!");
      return(FATAL);
    }

  while(!succeeded)
    {

      /* attempt to send the entire thing */
      bytesRead = read(getFromFD, &dataBuffer[*dataBytes],
                       bufferBytes - *dataBytes);

      /* the read got interrupted, retry */
      if(bytesRead == -1 && errno == EINTR)
        continue;

      /* did the read fail? */
      if(bytesRead < 0)
        {
          ERROR("GetData: failed reading from socket.");
          return(FATAL);
        }

      if(bytesRead == 0)
        {
          DEBUG("GetData: got EOF.");
          return(GOT_EOF);
        }

      if(bytesRead + *dataBytes > bufferBytes)
        {
          ERROR("GetData: buffer overrun.");
          return(FATAL);
        }

      succeeded = 1;
    }

  *dataBytes += bytesRead;

  DEBUG("GetData: got %ld bytes; now total in buffer: %ld bytes.",
```

```
        (long) bytesRead, *dataBytes);

  return(NO_ERR);
}


int SendData(int sendToFD, char *dataBuffer, long *dataBytes, long bufferBytes)
{
  ssize_t bytesWritten;
  int succeeded = 0;

  while(!succeeded)
    {
      /* attempt to send the entire thing */
      bytesWritten = write(sendToFD, dataBuffer, *dataBytes);

      /* the write got interrupted, retry */
      if(bytesWritten == -1 && errno == EINTR)
        continue;

      /* did the write fail? */
      if(bytesWritten <= 0)
        {
          ERROR("SendData: failed writing to socket.");
          return(FATAL);
        }

      if(bytesWritten > bufferBytes)
        {
          ERROR("SendData: buffer overrun.");
          return(FATAL);
        }

      succeeded = 1;
    }

  /* Pack the buffer if we didn't write the whole thing */
  if(bytesWritten < *dataBytes)
    {
      int i, j;

      DEBUG("SendData: packing buffer.");

      for(i = 0, j = bytesWritten; j < *dataBytes; i++, j++)
        dataBuffer[i] = dataBuffer[j];
    }

  *dataBytes -= bytesWritten;

  DEBUG("SendData: sent %ld bytes; now total in buffer: %ld bytes.",
        (long) bytesWritten, *dataBytes);

  return(NO_ERR);
}



int EGetData(int getFromFD, char *dataBuffer, long *dataBytes,
             long bufferBytes, Keys *getKeys)
```

```
{
  ssize_t       bytesRead;
  FourBytes     dataLength;
  FourBytes     actualDataLength;
  int           padding;
  unsigned long currentPtr;

  /* Sanity checks */
  if(*dataBytes >= bufferBytes)
    {
      ERROR("EGetData: buffer is already full!");
      return(FATAL);
    }

  /* Peek at the length byte */
  /* we don't want to grab it because we may want to abort if it won't fit */
  /* and we don't want to have to save more state */
  currentPtr = 0;
  while(currentPtr < sizeof(FourBytes))
    {
      bytesRead = recv(getFromFD, (char *)&dataLength + currentPtr,
                       sizeof(FourBytes) - currentPtr, MSG_PEEK);

      /* the recv got interrupted, retry */
      if(bytesRead == -1 && errno == EINTR)
        continue;

      /* did the recv fail? */
      if(bytesRead < 0)
        {
          ERROR("EGetData: recv failed reading from socket.");
          return(FATAL);
        }

      if(bytesRead == 0)
        {
          DEBUG("EGetData: got EOF.");
          return(GOT_EOF);
        }

      currentPtr += bytesRead;

      if(currentPtr > sizeof(FourBytes))
        {
          ERROR("EGetData: buffer overrun reading data length.");
          return(FATAL);
        }
    }

  /* convert to a byte order we understand */
  dataLength = ntohl(dataLength);

  /* figure out how much padding there will be on the message */
  padding = (8 - (dataLength % 8)) % 8;

  /* check to see if the length is valid */
  if((dataLength < 0) || (dataLength + padding > bufferBytes))
    {
      ERROR("EGetData: Invalid encrypted data length: %d.", dataLength);
```

```
    return(FATAL);
  }

dataLength += padding;

/* will the data fit in the buffer? */
if((*dataBytes + dataLength) > bufferBytes)
  {
    DEBUG("EGetData: only %ld bytes free in dataBuffer.",
          bufferBytes - *dataBytes);
    return(FULL);
  }

/* We have room for the data, read in the length byte for real */
currentPtr = 0;
while(currentPtr < sizeof(FourBytes))
  {
    bytesRead = read(getFromFD, &actualDataLength + currentPtr,
                     sizeof(FourBytes) - currentPtr);

    /* the read got interrupted, retry */
    if(bytesRead == -1 && errno == EINTR)
      continue;

    /* did the read fail? */
    if(bytesRead < 0)
      {
        ERROR("EGetData: failed reading from socket.");
        return(FATAL);
      }

    if(bytesRead == 0)
      {
        DEBUG("GetData: got EOF.");
        return(GOT_EOF);
      }

    currentPtr += bytesRead;

    if(currentPtr > sizeof(FourBytes))
      {
        ERROR("EGetData: buffer overrun reading data length.");
        return(FATAL);
      }

  }

/* convert to a byte order we understand */
actualDataLength = ntohl(actualDataLength);

/* sanity check to make sure we get the same length */
if((actualDataLength + padding) != dataLength)
  {
    ERROR("EGetData: data length byte changed!!");
    return(FATAL);
  }

/* now read in the actual data */
currentPtr = 0;
```

```
  while(currentPtr < dataLength)
    {
      /* attempt to read the entire message */
      bytesRead = read(getFromFD, &dataBuffer[*dataBytes + currentPtr],
                       dataLength - currentPtr);

      /* the read got interrupted, retry */
      if(bytesRead == -1 && errno == EINTR)
        continue;

      /* did the read fail? */
      if(bytesRead < 0)
        {
          ERROR("EGetData: failed reading from socket.");
          return(FATAL);
        }

      if(bytesRead == 0)
        {
          ERROR("EGetData: got EOF while trying to read message.");
          return(FATAL);
        }

      currentPtr += bytesRead;

      if((*dataBytes + currentPtr) > bufferBytes)
        {
          ERROR("EGetData: anticipated buffer overrun.");
          return(FATAL);
        }
    }

  /* decrypt the data */
  des_3cbc_decrypt(&getKeys->key1, getKeys->ivec1,
                   &getKeys->key2, getKeys->ivec2,
                   &getKeys->key3, getKeys->ivec3,
                   &dataBuffer[*dataBytes], &dataBuffer[*dataBytes],
                   dataLength);

  *dataBytes += actualDataLength;

  DEBUG("EGetData: got %d actual bytes (%d encrypted); total: %ld bytes.",
        actualDataLength, dataLength, *dataBytes);

  return(NO_ERR);
}



int ESendData(int sendToFD, char *dataBuffer, long *dataBytes,
              long bufferBytes, Keys *sendKeys)
{
  ssize_t       bytesWritten;
  long          currentPtr;
  FourBytes     dataLength;
  int           padding;

  /* write out the whole buffer */
  dataLength = htonl(*dataBytes);
```

```
currentPtr = 0;
while(currentPtr < sizeof(FourBytes))
  {
    /* attempt to send the entire thing */
    bytesWritten = write(sendToFD, (&dataLength) + currentPtr,
                         sizeof(FourBytes) - currentPtr);

    /* the write got interrupted, retry */
    if(bytesWritten == -1 && errno == EINTR)
      continue;

    /* did the write fail? */
    if(bytesWritten <= 0)
      {
        ERROR("SendData: failed writing to socket.");
        return(FATAL);
      }

    currentPtr += bytesWritten;

    if(currentPtr > sizeof(FourBytes))
      {
        ERROR("ESendData: overrun writing data length.");
        return(FATAL);
      }
  }

padding = (8 - (*dataBytes % 8)) % 8;
DEBUG("ESendData: data was %ld bytes, padding by %d bytes.",
      *dataBytes, padding);

/* pad with 0s */
memset(&dataBuffer[*dataBytes], 0, padding);

*dataBytes += padding;

/* Now we encrypt the data and send it */
des_3cbc_encrypt(&sendKeys->key1, sendKeys->ivec1,
                 &sendKeys->key2, sendKeys->ivec2,
                 &sendKeys->key3, sendKeys->ivec3,
                 dataBuffer, dataBuffer, *dataBytes);

/* send the encrypted data */
currentPtr = 0;
while(currentPtr < *dataBytes)
  {
    /* attempt to write the entire message */
    bytesWritten = write(sendToFD, &dataBuffer[currentPtr],
                         *dataBytes - currentPtr);

    /* the write got interrupted, retry */
    if(bytesWritten == -1 && errno == EINTR)
      continue;

    /* did the write fail? */
    if(bytesWritten < 0)
      {
        ERROR("ESendData: failed writing to socket.");
```

```
            return(FATAL);
          }

        if(bytesWritten == 0)
          {
            ERROR("ESendData: got EOF while trying to send message.");
            return(FATAL);
          }

        currentPtr += bytesWritten;

        if(currentPtr > bufferBytes)
          {
            ERROR("ESendData: buffer overrun.");
            return(FATAL);
          }
      }

    *dataBytes = 0;

    return(NO_ERR);
}
```

File Name: des.c

```
/*

DES implementation; 1995 Tatu Ylonen <ylo@cs.hut.fi>

This implementation is derived from libdes-3.06, which is copyright
(c) 1993 Eric Young, and distributed under the GNU GPL or the ARTISTIC licence
(at the user's option).  The original distribution can be found e.g. from
ftp://ftp.dsi.unimi.it/pub/security/crypt/libdes/libdes-3.06.tar.gz.

This implementation is distributed under the same terms.  See
libdes-README, libdes-ARTISTIC, and libdes-COPYING for more
information.

A description of the DES algorithm can be found in every modern book on
cryptography and data security, including the following:

  Bruce Schneier: Applied Cryptography.  John Wiley & Sons, 1994.

  Jennifer Seberry and Josed Pieprzyk: Cryptography: An Introduction to
    Computer Security.  Prentice-Hall, 1989.

  Man Young Rhee: Cryptography and Secure Data Communications.  McGraw-Hill,
    1994.

*/

/*
 * $Id: des.c,v 1.1.1.1 1996/02/18 21:38:11 ylo Exp $
 * $Log: des.c,v $
 * Revision 1.1.1.1  1996/02/18 21:38:11  ylo
 *      Imported ssh-1.2.13.
 *
 * Revision 1.2  1995/07/13  01:22:25  ylo
 *      Added cvs log.
 *
 * $Endlog$
 */

#include "shared.h"
#include "getput.h"

/* Table for key generation.  This used to be in sk.h. */
/* Copyright (C) 1993 Eric Young - see README for more details */
static const word32 des_skb[8][64]={
/* for C bits (numbered as per FIPS 46) 1 2 3 4 5 6 */
{ 0x00000000,0x00000010,0x20000000,0x20000010,
0x00010000,0x00010010,0x20010000,0x20010010,
0x00000800,0x00000810,0x20000800,0x20000810,
0x00010800,0x00010810,0x20010800,0x20010810,
0x00000020,0x00000030,0x20000020,0x20000030,
0x00010020,0x00010030,0x20010020,0x20010030,
0x00000820,0x00000830,0x20000820,0x20000830,
0x00010820,0x00010830,0x20010820,0x20010830,
0x00080000,0x00080010,0x20080000,0x20080010,
0x00090000,0x00090010,0x20090000,0x20090010,
0x00080800,0x00080810,0x20080800,0x20080810,
```

```
0x00090800,0x00090810,0x20090800,0x20090810,
0x00080020,0x00080030,0x20080020,0x20080030,
0x00090020,0x00090030,0x20090020,0x20090030,
0x00080820,0x00080830,0x20080820,0x20080830,
0x00090820,0x00090830,0x20090820,0x20090830 },
/* for C bits (numbered as per FIPS 46) 7 8 10 11 12 13 */
{ 0x00000000,0x02000000,0x00002000,0x02002000,
0x00200000,0x02200000,0x00202000,0x02202000,
0x00000004,0x02000004,0x00002004,0x02002004,
0x00200004,0x02200004,0x00202004,0x02202004,
0x00000400,0x02000400,0x00002400,0x02002400,
0x00200400,0x02200400,0x00202400,0x02202400,
0x00000404,0x02000404,0x00002404,0x02002404,
0x00200404,0x02200404,0x00202404,0x02202404,
0x10000000,0x12000000,0x10002000,0x12002000,
0x10200000,0x12200000,0x10202000,0x12202000,
0x10000004,0x12000004,0x10002004,0x12002004,
0x10200004,0x12200004,0x10202004,0x12202004,
0x10000400,0x12000400,0x10002400,0x12002400,
0x10200400,0x12200400,0x10202400,0x12202400,
0x10000404,0x12000404,0x10002404,0x12002404,
0x10200404,0x12200404,0x10202404,0x12202404 },
/* for C bits (numbered as per FIPS 46) 14 15 16 17 19 20 */
{ 0x00000000,0x00000001,0x00040000,0x00040001,
0x01000000,0x01000001,0x01040000,0x01040001,
0x00000002,0x00000003,0x00040002,0x00040003,
0x01000002,0x01000003,0x01040002,0x01040003,
0x00000200,0x00000201,0x00040200,0x00040201,
0x01000200,0x01000201,0x01040200,0x01040201,
0x00000202,0x00000203,0x00040202,0x00040203,
0x01000202,0x01000203,0x01040202,0x01040203,
0x08000000,0x08000001,0x08040000,0x08040001,
0x09000000,0x09000001,0x09040000,0x09040001,
0x08000002,0x08000003,0x08040002,0x08040003,
0x09000002,0x09000003,0x09040002,0x09040003,
0x08000200,0x08000201,0x08040200,0x08040201,
0x09000200,0x09000201,0x09040200,0x09040201,
0x08000202,0x08000203,0x08040202,0x08040203,
0x09000202,0x09000203,0x09040202,0x09040203 },
/* for C bits (numbered as per FIPS 46) 21 23 24 26 27 28 */
{ 0x00000000,0x00100000,0x00000100,0x00100100,
0x00000008,0x00100008,0x00000108,0x00100108,
0x00001000,0x00101000,0x00001100,0x00101100,
0x00001008,0x00101008,0x00001108,0x00101108,
0x04000000,0x04100000,0x04000100,0x04100100,
0x04000008,0x04100008,0x04000108,0x04100108,
0x04001000,0x04101000,0x04001100,0x04101100,
0x04001008,0x04101008,0x04001108,0x04101108,
0x00020000,0x00120000,0x00020100,0x00120100,
0x00020008,0x00120008,0x00020108,0x00120108,
0x00021000,0x00121000,0x00021100,0x00121100,
0x00021008,0x00121008,0x00021108,0x00121108,
0x04020000,0x04120000,0x04020100,0x04120100,
0x04020008,0x04120008,0x04020108,0x04120108,
0x04021000,0x04121000,0x04021100,0x04121100,
0x04021008,0x04121008,0x04021108,0x04121108 },
/* for D bits (numbered as per FIPS 46) 1 2 3 4 5 6 */
{ 0x00000000,0x10000000,0x00010000,0x10010000,
0x00000004,0x10000004,0x00010004,0x10010004,
```

```
0x20000000,0x30000000,0x20010000,0x30010000,
0x20000004,0x30000004,0x20010004,0x30010004,
0x00100000,0x10100000,0x00110000,0x10110000,
0x00100004,0x10100004,0x00110004,0x10110004,
0x20100000,0x30100000,0x20110000,0x30110000,
0x20100004,0x30100004,0x20110004,0x30110004,
0x00001000,0x10001000,0x00011000,0x10011000,
0x00001004,0x10001004,0x00011004,0x10011004,
0x20001000,0x30001000,0x20011000,0x30011000,
0x20001004,0x30001004,0x20011004,0x30011004,
0x00101000,0x10101000,0x00111000,0x10111000,
0x00101004,0x10101004,0x00111004,0x10111004,
0x20101000,0x30101000,0x20111000,0x30111000,
0x20101004,0x30101004,0x20111004,0x30111004 },
/* for D bits (numbered as per FIPS 46) 8 9 11 12 13 14 */
{ 0x00000000,0x08000000,0x00000008,0x08000008,
0x00000400,0x08000400,0x00000408,0x08000408,
0x00020000,0x08020000,0x00020008,0x08020008,
0x00020400,0x08020400,0x00020408,0x08020408,
0x00000001,0x08000001,0x00000009,0x08000009,
0x00000401,0x08000401,0x00000409,0x08000409,
0x00020001,0x08020001,0x00020009,0x08020009,
0x00020401,0x08020401,0x00020409,0x08020409,
0x02000000,0x0A000000,0x02000008,0x0A000008,
0x02000400,0x0A000400,0x02000408,0x0A000408,
0x02020000,0x0A020000,0x02020008,0x0A020008,
0x02020400,0x0A020400,0x02020408,0x0A020408,
0x02000001,0x0A000001,0x02000009,0x0A000009,
0x02000401,0x0A000401,0x02000409,0x0A000409,
0x02020001,0x0A020001,0x02020009,0x0A020009,
0x02020401,0x0A020401,0x02020409,0x0A020409 },
/* for D bits (numbered as per FIPS 46) 16 17 18 19 20 21 */
{ 0x00000000,0x00000100,0x00080000,0x00080100,
0x01000000,0x01000100,0x01080000,0x01080100,
0x00000010,0x00000110,0x00080010,0x00080110,
0x01000010,0x01000110,0x01080010,0x01080110,
0x00200000,0x00200100,0x00280000,0x00280100,
0x01200000,0x01200100,0x01280000,0x01280100,
0x00200010,0x00200110,0x00280010,0x00280110,
0x01200010,0x01200110,0x01280010,0x01280110,
0x00000200,0x00000300,0x00080200,0x00080300,
0x01000200,0x01000300,0x01080200,0x01080300,
0x00000210,0x00000310,0x00080210,0x00080310,
0x01000210,0x01000310,0x01080210,0x01080310,
0x00200200,0x00200300,0x00280200,0x00280300,
0x01200200,0x01200300,0x01280200,0x01280300,
0x00200210,0x00200310,0x00280210,0x00280310,
0x01200210,0x01200310,0x01280210,0x01280310 },
/* for D bits (numbered as per FIPS 46) 22 23 24 25 27 28 */
{ 0x00000000,0x04000000,0x00040000,0x04040000,
0x00000002,0x04000002,0x00040002,0x04040002,
0x00002000,0x04002000,0x00042000,0x04042000,
0x00002002,0x04002002,0x00042002,0x04042002,
0x00000020,0x04000020,0x00040020,0x04040020,
0x00000022,0x04000022,0x00040022,0x04040022,
0x00002020,0x04002020,0x00042020,0x04042020,
0x00002022,0x04002022,0x00042022,0x04042022,
0x00000800,0x04000800,0x00040800,0x04040800,
0x00000802,0x04000802,0x00040802,0x04040802,
```

```
0x00002800,0x04002800,0x00042800,0x04042800,
0x00002802,0x04002802,0x00042802,0x04042802,
0x00000820,0x04000820,0x00040820,0x04040820,
0x00000822,0x04000822,0x00040822,0x04040822,
0x00002820,0x04002820,0x00042820,0x04042820,
0x00002822,0x04002822,0x00042822,0x04042822 }
};

/* Tables used for executing des.  This used to be in spr.h. */
/* Copyright (C) 1993 Eric Young - see README for more details */
static const word32 des_SPtrans[8][64]={
/* nibble 0 */
{ 0x00820200, 0x00020000, 0x80800000, 0x80820200,
0x00800000, 0x80020200, 0x80020000, 0x80800000,
0x80020200, 0x00820200, 0x00820000, 0x80000200,
0x80800200, 0x00800000, 0x00000000, 0x80020000,
0x00020000, 0x80000000, 0x00800200, 0x00020200,
0x80820200, 0x00820000, 0x80000200, 0x00800200,
0x80000000, 0x00000200, 0x00020200, 0x80820000,
0x00000200, 0x80800200, 0x80820000, 0x00000000,
0x00000000, 0x80820200, 0x00800200, 0x80020000,
0x00820200, 0x00020000, 0x80000200, 0x00800200,
0x80820000, 0x00000200, 0x00020200, 0x80800000,
0x80020200, 0x80000000, 0x80800000, 0x00820000,
0x80820200, 0x00020200, 0x00820000, 0x80800200,
0x00800000, 0x80000200, 0x80020000, 0x00000000,
0x00020000, 0x00800000, 0x80800200, 0x00820200,
0x80000000, 0x80820000, 0x00000200, 0x80020200 },

/* nibble 1 */
{ 0x10042004, 0x00000000, 0x00042000, 0x10040000,
0x10000004, 0x00002004, 0x10002000, 0x00042000,
0x00002000, 0x10040004, 0x00000004, 0x10002000,
0x00040004, 0x10042000, 0x10040000, 0x00000004,
0x00040000, 0x10002004, 0x10040004, 0x00002000,
0x00042004, 0x10000000, 0x00000000, 0x00040004,
0x10002004, 0x00042004, 0x10042000, 0x10000004,
0x10000000, 0x00040000, 0x00002004, 0x10042004,
0x00040004, 0x10042000, 0x10002000, 0x00042004,
0x10042004, 0x00040004, 0x10000004, 0x00000000,
0x10000000, 0x00002004, 0x00040000, 0x10040004,
0x00002000, 0x10000000, 0x00042004, 0x10002004,
0x10042000, 0x00002000, 0x00000000, 0x10000004,
0x00000004, 0x10042004, 0x00042000, 0x10040000,
0x10040004, 0x00040000, 0x00002004, 0x10002000,
0x10002004, 0x00000004, 0x10040000, 0x00042000 },

/* nibble 2 */
{ 0x41000000, 0x01010040, 0x00000040, 0x41000040,
0x40010000, 0x01000000, 0x41000040, 0x00010040,
0x01000040, 0x00010000, 0x01010000, 0x40000000,
0x41010040, 0x40000040, 0x40000000, 0x41010000,
0x00000000, 0x40010000, 0x01010040, 0x00000040,
0x40000040, 0x41010040, 0x00010000, 0x41000000,
0x41010000, 0x01000040, 0x40010040, 0x01010000,
0x00010040, 0x00000000, 0x01000000, 0x40010040,
0x01010040, 0x00000040, 0x40000000, 0x00010000,
0x40000040, 0x40010000, 0x01010000, 0x41000040,
0x00000000, 0x01010040, 0x00010040, 0x41010000,
```

```
0x40010000, 0x01000000, 0x41010040, 0x40000000,
0x40010040, 0x41000000, 0x01000000, 0x41010040,
0x00010000, 0x01000040, 0x41000040, 0x00010040,
0x01000040, 0x00000000, 0x41010000, 0x40000040,
0x41000000, 0x40010040, 0x00000040, 0x01010000 },

/* nibble 3 */
{ 0x00100402, 0x04000400, 0x00000002, 0x04100402,
0x00000000, 0x04100000, 0x04000402, 0x00100002,
0x04100400, 0x04000002, 0x04000000, 0x00000402,
0x04000002, 0x00100402, 0x00100000, 0x04000000,
0x04100002, 0x00100400, 0x00000400, 0x00000002,
0x00100400, 0x04000402, 0x04100000, 0x00000400,
0x00000402, 0x00000000, 0x00100002, 0x04100400,
0x04000400, 0x04100002, 0x04100402, 0x00100000,
0x04100002, 0x00000402, 0x00100000, 0x04000002,
0x00100400, 0x04000400, 0x00000002, 0x04100000,
0x04000402, 0x00000000, 0x00000400, 0x00100002,
0x00000000, 0x04100002, 0x04100400, 0x00000400,
0x04000000, 0x04100402, 0x00100402, 0x00100000,
0x04100402, 0x00000002, 0x04000400, 0x00100402,
0x00100002, 0x00100400, 0x04100000, 0x04000402,
0x00000402, 0x04000000, 0x04000002, 0x04100400 },

/* nibble 4 */
{ 0x02000000, 0x00004000, 0x00000100, 0x02004108,
0x02004008, 0x02000100, 0x00004108, 0x02004000,
0x00004000, 0x00000008, 0x02000008, 0x00004100,
0x02000108, 0x02004008, 0x02004100, 0x00000000,
0x00004100, 0x02000000, 0x00004008, 0x00000108,
0x02000100, 0x00004108, 0x00000000, 0x02000008,
0x00000008, 0x02000108, 0x02004108, 0x00004008,
0x02004000, 0x00000100, 0x00000108, 0x02004100,
0x02004100, 0x02000108, 0x00004008, 0x02004000,
0x00004000, 0x00000008, 0x02000008, 0x02000100,
0x02000000, 0x00004100, 0x02004108, 0x00000000,
0x00004108, 0x02000000, 0x00000100, 0x00004008,
0x02000108, 0x00000100, 0x00000000, 0x02004108,
0x02004008, 0x02004100, 0x00000108, 0x00004000,
0x00004100, 0x02004008, 0x02000100, 0x00000108,
0x00000008, 0x00004108, 0x02004000, 0x02000008 },

/* nibble 5 */
{ 0x20000010, 0x00080010, 0x00000000, 0x20080800,
0x00080010, 0x00000800, 0x20000810, 0x00080000,
0x00000810, 0x20080810, 0x00080800, 0x20000000,
0x20000800, 0x20000010, 0x20080000, 0x00080810,
0x00080000, 0x20000810, 0x20080010, 0x00000000,
0x00000800, 0x00000010, 0x20080800, 0x20080010,
0x20080810, 0x20080000, 0x20000000, 0x00000810,
0x00000010, 0x00080800, 0x00080810, 0x20000800,
0x00000810, 0x20000000, 0x20000800, 0x00080810,
0x20080800, 0x00080010, 0x00000000, 0x20000800,
0x20000000, 0x00000800, 0x20080010, 0x00080000,
0x00080010, 0x20080810, 0x00080800, 0x00000010,
0x20080810, 0x00080800, 0x00080000, 0x20000810,
0x20000010, 0x20080000, 0x00080810, 0x00000000,
0x00000800, 0x20000010, 0x20000810, 0x20080800,
0x20080000, 0x00000810, 0x00000010, 0x20080010 },
```

```
/* nibble 6 */
{ 0x00001000, 0x00000080, 0x00400080, 0x00400001,
0x00401081, 0x00001001, 0x00001080, 0x00000000,
0x00400000, 0x00400081, 0x00000081, 0x00401000,
0x00000001, 0x00401080, 0x00401000, 0x00000081,
0x00400081, 0x00001000, 0x00001001, 0x00401081,
0x00000000, 0x00400080, 0x00400001, 0x00001080,
0x00401001, 0x00001081, 0x00401080, 0x00000001,
0x00001081, 0x00401001, 0x00000080, 0x00400000,
0x00001081, 0x00401000, 0x00401001, 0x00000081,
0x00001000, 0x00000080, 0x00400000, 0x00401001,
0x00400081, 0x00001081, 0x00001080, 0x00000000,
0x00000080, 0x00400001, 0x00000001, 0x00400080,
0x00000000, 0x00400081, 0x00400080, 0x00001080,
0x00000081, 0x00001000, 0x00401081, 0x00400000,
0x00401080, 0x00000001, 0x00001001, 0x00401081,
0x00400001, 0x00401080, 0x00401000, 0x00001001 },

/* nibble 7 */
{ 0x08200020, 0x08208000, 0x00008020, 0x00000000,
0x08008000, 0x00200020, 0x08200000, 0x08208020,
0x00000020, 0x08000000, 0x00208000, 0x00008020,
0x00208020, 0x08008020, 0x08000020, 0x08200000,
0x00008000, 0x00208020, 0x00200020, 0x08008000,
0x08208020, 0x08000020, 0x00000000, 0x00208000,
0x08000000, 0x00200000, 0x08008020, 0x08200020,
0x00200000, 0x00008000, 0x08208000, 0x00000020,
0x00200000, 0x00008000, 0x08000020, 0x08208020,
0x00008020, 0x08000000, 0x00000000, 0x00208000,
0x08200020, 0x08008020, 0x08008000, 0x00200020,
0x08208000, 0x00000020, 0x00200020, 0x08008000,
0x08208020, 0x00200000, 0x08200000, 0x08000020,
0x00208000, 0x00008020, 0x08008020, 0x08200000,
0x00000020, 0x08208000, 0x00208020, 0x00000000,
0x08000000, 0x08200020, 0x00008000, 0x00208020 }};

/* Some stuff that used to be in des_locl.h.  Heavily modified. */
        /* IP and FP
         * The problem is more of a geometric problem that random bit fiddling.
         0  1  2  3  4  5  6  7      62 54 46 38 30 22 14  6
         8  9 10 11 12 13 14 15      60 52 44 36 28 20 12  4
        16 17 18 19 20 21 22 23      58 50 42 34 26 18 10  2
        24 25 26 27 28 29 30 31  to  56 48 40 32 24 16  8  0

        32 33 34 35 36 37 38 39      63 55 47 39 31 23 15  7
        40 41 42 43 44 45 46 47      61 53 45 37 29 21 13  5
        48 49 50 51 52 53 54 55      59 51 43 35 27 19 11  3
        56 57 58 59 60 61 62 63      57 49 41 33 25 17  9  1

        The output has been subject to swaps of the form
        0 1 -> 3 1 but the odd and even bits have been put into
        2 3    2 0
        different words.  The main trick is to remember that
        t=((l>>size)^r)&(mask);
        r^=t;
        l^=(t<<size);
        can be used to swap and move bits between words.
```

```
        So l =  0  1  2  3  r = 16 17 18 19
                4  5  6  7      20 21 22 23
                8  9 10 11      24 25 26 27
               12 13 14 15      28 29 30 31
        becomes (for size == 2 and mask == 0x3333)
           t =   2^16  3^17 -- --   l =  0  1 16 17  r =  2  3 18 19
                 6^20  7^21 -- --        4  5 20 21       6  7 22 23
                10^24 11^25 -- --        8  9 24 25      10 11 24 25
                14^28 15^29 -- --       12 13 28 29      14 15 28 29

        Thanks for hints from Richard Outerbridge - he told me IP&FP
        could be done in 15 xor, 10 shifts and 5 ands.
        When I finally started to think of the problem in 2D
        I first got ~42 operations without xors.  When I remembered
        how to use xors :-) I got it to its final state.
        */
#define PERM_OP(a,b,t,n,m) ((t)=((((a)>>(n))^(b))&(m)),\
        (b)^=(t),\
        (a)^=((t)<<(n)))

#define IP(l,r,t) \
        PERM_OP(r,l,t, 4,0x0f0f0f0f); \
        PERM_OP(l,r,t,16,0x0000ffff); \
        PERM_OP(r,l,t, 2,0x33333333); \
        PERM_OP(l,r,t, 8,0x00ff00ff); \
        PERM_OP(r,l,t, 1,0x55555555);

#define FP(l,r,t) \
        PERM_OP(l,r,t, 1,0x55555555); \
        PERM_OP(r,l,t, 8,0x00ff00ff); \
        PERM_OP(l,r,t, 2,0x33333333); \
        PERM_OP(r,l,t,16,0x0000ffff); \
        PERM_OP(l,r,t, 4,0x0f0f0f0f);

#define D_ENCRYPT(L,R,S)        \
        u=(R^s[S  ]); \
        t=R^s[S+1]; \
        t=((t>>4)+(t<<28)); \
        L^=     des_SPtrans[1][(t    )&0x3f]| \
                des_SPtrans[3][(t>> 8)&0x3f]| \
                des_SPtrans[5][(t>>16)&0x3f]| \
                des_SPtrans[7][(t>>24)&0x3f]| \
                des_SPtrans[0][(u    )&0x3f]| \
                des_SPtrans[2][(u>> 8)&0x3f]| \
                des_SPtrans[4][(u>>16)&0x3f]| \
                des_SPtrans[6][(u>>24)&0x3f];

/* This part is based on code that used to be in ecb_enc.c. */
/* Copyright (C) 1993 Eric Young - see README for more details */

void des_encrypt(word32 l, word32 r, word32 *output, DESContext *ks,
                 int encrypt)
{
  register word32 t,u;
  register int i;
  register word32 *s;

  s = ks->key_schedule;
```

```
  IP(l,r,t);
  /* Things have been modified so that the initial rotate is
   * done outside the loop.  This required the
   * des_SPtrans values in sp.h to be rotated 1 bit to the right.
   * One perl script later and things have a 5% speed up on a sparc2.
   * Thanks to Richard Outerbridge <71755.204@CompuServe.COM>
   * for pointing this out. */
  t=(r<<1)|(r>>31);
  r=(l<<1)|(l>>31);
  l=t;

  /* I don't know if it is worth the effort of loop unrolling the
   * inner loop */
  if (encrypt)
    {
      for (i=0; i<32; i+=4)
        {
          D_ENCRYPT(l,r,i+0); /*  1 */
          D_ENCRYPT(r,l,i+2); /*  2 */
        }
    }
  else
    {
      for (i=30; i>0; i-=4)
        {
          D_ENCRYPT(l,r,i-0); /* 16 */
          D_ENCRYPT(r,l,i-2); /* 15 */
        }
    }
  l=(l>>1)|(l<<31);
  r=(r>>1)|(r<<31);

  FP(r,l,t);
  output[0]=l;
  output[1]=r;
}

/* Code based on set_key.c. */
/* Copyright (C) 1993 Eric Young - see README for more details */

#define HPERM_OP(a,t,n,m) ((t)=((((a)<<(16-(n)))^(a))&(m)),\
        (a)=(a)^(t)^(t>>(16-(n))))

void des_set_key(unsigned char *key, DESContext *ks)
{
  register word32 c, d, t, s, shifts;
  register int i;
  register word32 *schedule;

  schedule = ks->key_schedule;

  c = GET_32BIT_LSB_FIRST(key);
  d = GET_32BIT_LSB_FIRST(key + 4);

  /* I now do it in 47 simple operations :-)
   * Thanks to John Fletcher (john_fletcher@lccmail.ocf.llnl.gov)
   * for the inspiration. :-) */
  PERM_OP(d,c,t,4,0x0f0f0f0f);
  HPERM_OP(c,t,-2,0xcccc0000);
```

```
  HPERM_OP(d,t,-2,0xcccc0000);
  PERM_OP(d,c,t,1,0x55555555);
  PERM_OP(c,d,t,8,0x00ff00ff);
  PERM_OP(d,c,t,1,0x55555555);
  d = ((d & 0xff) << 16) | (d & 0xff00) |
    ((d >> 16) & 0xff) | ((c >> 4) & 0xf000000);
  c&=0x0fffffff;

  shifts = 0x7efc;
  for (i=0; i < 16; i++)
    {
      if (shifts & 1)
        { c=((c>>2)|(c<<26)); d=((d>>2)|(d<<26)); }
      else
        { c=((c>>1)|(c<<27)); d=((d>>1)|(d<<27)); }
      shifts >>= 1;
      c&=0x0fffffff;
      d&=0x0fffffff;

      /* could be a few less shifts but I am to lazy at this
       * point in time to investigate */

      s = des_skb[0][ (c    )&0x3f                ] |
          des_skb[1][((c>> 6)&0x03)|((c>> 7)&0x3c)] |
          des_skb[2][((c>>13)&0x0f)|((c>>14)&0x30)] |
          des_skb[3][((c>>20)&0x01)|((c>>21)&0x06)|((c>>22)&0x38)];

      t = des_skb[4][ (d    )&0x3f                ] |
          des_skb[5][((d>> 7)&0x03)|((d>> 8)&0x3c)] |
          des_skb[6][ (d>>15)&0x3f                ] |
          des_skb[7][((d>>21)&0x0f)|((d>>22)&0x30)];

      /* table contained 0213 4657 */
      *schedule++ = ((t << 16) | (s & 0xffff));
      s = ((s >> 16) | (t & 0xffff0000));
      *schedule++ = (s << 4) | (s >> 28);
    }
}

void des_cbc_encrypt(DESContext *ks, unsigned char *iv,
                     unsigned char *dest, const unsigned char *src,
                     unsigned int len)
{
  word32 iv0, iv1, out[2];
  unsigned int i;

  assert((len & 7) == 0);

  iv0 = GET_32BIT_LSB_FIRST(iv);
  iv1 = GET_32BIT_LSB_FIRST(iv + 4);

  for (i = 0; i < len; i += 8)
    {
      iv0 ^= GET_32BIT_LSB_FIRST(src + i);
      iv1 ^= GET_32BIT_LSB_FIRST(src + i + 4);
      des_encrypt(iv0, iv1, out, ks, 1);
      iv0 = out[0];
      iv1 = out[1];
      PUT_32BIT_LSB_FIRST(dest + i, iv0);
```

```
      PUT_32BIT_LSB_FIRST(dest + i + 4, iv1);
    }
  PUT_32BIT_LSB_FIRST(iv, iv0);
  PUT_32BIT_LSB_FIRST(iv + 4, iv1);
}

void des_cbc_decrypt(DESContext *ks, unsigned char *iv,
                     unsigned char *dest, const unsigned char *src,
                     unsigned int len)
{
  word32 iv0, iv1, d0, d1, out[2];
  unsigned int i;

  assert((len & 7) == 0);

  iv0 = GET_32BIT_LSB_FIRST(iv);
  iv1 = GET_32BIT_LSB_FIRST(iv + 4);

  for (i = 0; i < len; i += 8)
    {
      d0 = GET_32BIT_LSB_FIRST(src + i);
      d1 = GET_32BIT_LSB_FIRST(src + i + 4);
      des_encrypt(d0, d1, out, ks, 0);
      iv0 ^= out[0];
      iv1 ^= out[1];
      PUT_32BIT_LSB_FIRST(dest + i, iv0);
      PUT_32BIT_LSB_FIRST(dest + i + 4, iv1);
      iv0 = d0;
      iv1 = d1;
    }
  PUT_32BIT_LSB_FIRST(iv, iv0);
  PUT_32BIT_LSB_FIRST(iv + 4, iv1);
}

void des_3cbc_encrypt(DESContext *ks1, unsigned char *iv1,
                      DESContext *ks2, unsigned char *iv2,
                      DESContext *ks3, unsigned char *iv3,
                      unsigned char *dest, const unsigned char *src,
                      unsigned int len)
{
  des_cbc_encrypt(ks1, iv1, dest, src, len);
  des_cbc_decrypt(ks2, iv2, dest, dest, len);
  des_cbc_encrypt(ks3, iv3, dest, dest, len);
}

void des_3cbc_decrypt(DESContext *ks1, unsigned char *iv1,
                      DESContext *ks2, unsigned char *iv2,
                      DESContext *ks3, unsigned char *iv3,
                      unsigned char *dest, const unsigned char *src,
                      unsigned int len)
{
  des_cbc_decrypt(ks3, iv3, dest, src, len);
  des_cbc_encrypt(ks2, iv2, dest, dest, len);
  des_cbc_decrypt(ks1, iv1, dest, dest, len);
}

#ifdef DES_TEST

void des_encrypt_buf(DESContext *ks, unsigned char *out,
```

```
                    const unsigned char *in, int encrypt)
{
  word32 in0, in1, output[0];

  in0 = GET_32BIT_LSB_FIRST(in);
  in1 = GET_32BIT_LSB_FIRST(in + 4);
  des_encrypt(in0, in1, output, ks, encrypt);
  PUT_32BIT_LSB_FIRST(out, output[0]);
  PUT_32BIT_LSB_FIRST(out + 4, output[1]);
}

int main(int ac, char **av)
{
  FILE *f;
  char line[1024], *cp;
  int i, value;
  unsigned char key[8], data[8], result[8], output[8];
  DESContext ks;

  while (fgets(line, sizeof(line), stdin))
    {
      for (i = 0; i < 8; i++)
        {
          if (sscanf(line + 2 * i, "%02x", &value) != 1)
            {
              fprintf(stderr, "1st col, i = %d, line: %s", i, line);
              exit(1);
            }
          key[i] = value;
        }
      for (i = 0; i < 8; i++)
        {
          if (sscanf(line + 2 * i + 17, "%02x", &value) != 1)
            {
              fprintf(stderr, "2nd col, i = %d, line: %s", i, line);
              exit(1);
            }
          data[i] = value;
        }
      for (i = 0; i < 8; i++)
        {
          if (sscanf(line + 2 * i + 2*17, "%02x", &value) != 1)
            {
              fprintf(stderr, "3rd col, i = %d, line: %s", i, line);
              exit(1);
            }
          result[i] = value;
        }
      des_set_key(key, &ks);
      des_encrypt_buf(&ks, output, data, 1);
      if (memcmp(output, result, 8) != 0)
        fprintf(stderr, "Encrypt failed: %s", line);
      des_encrypt_buf(&ks, output, result, 0);
      if (memcmp(output, data, 8) != 0)
        fprintf(stderr, "Decrypt failed: %s", line);
    }
  exit(0);
}
#endif /* DES_TEST */
```

File Name: des.h

```
/*

DES implementation; 1995 Tatu Ylonen <ylo@cs.hut.fi>

This implementation is derived from libdes-3.06, which is copyright
(c) 1993 Eric Young, and distributed under the GNU GPL or the ARTISTIC licence
(at the user's option).  The original distribution can be found e.g. from
ftp://ftp.dsi.unimi.it/pub/security/crypt/libdes/libdes-3.06.tar.gz.

This implementation is distributed under the same terms.  See
libdes-README, libdes-ARTISTIC, and libdes-COPYING for more
information.

*/

/*
 * $Id: des.h,v 1.1.1.1 1996/02/18 21:38:11 ylo Exp $
 * $Log: des.h,v $
 * Revision 1.1.1.1  1996/02/18 21:38:11  ylo
 *      Imported ssh-1.2.13.
 *
 * Revision 1.2  1995/07/13  01:22:57  ylo
 *      Added cvs log.
 *
 * $Endlog$
 */

#ifndef DES_H
#define DES_H

typedef unsigned int word32;

typedef struct
{
  word32 key_schedule[32];
} DESContext;

/* Sets the des key for the context.  Initializes the context.  The least
   significant bit of each byte of the key is ignored as parity. */
void des_set_key(unsigned char *key, DESContext *ks);

/* Encrypts 32 bits in l,r, and stores the result in output[0] and output[1].
   Performs encryption if encrypt is non-zero, and decryption if it is zero.
   The key context must have been initialized previously with des_set_key. */
void des_encrypt(word32 l, word32 r, word32 *output, DESContext *ks,
                 int encrypt);

/* Encrypts len bytes from src to dest in CBC modes.  Len must be a multiple
   of 8.  iv will be modified at end to a value suitable for continuing
   encryption. */
void des_cbc_encrypt(DESContext *ks, unsigned char *iv, unsigned char *dest,
                     const unsigned char *src, unsigned int len);

/* Decrypts len bytes from src to dest in CBC modes.  Len must be a multiple
   of 8.  iv will be modified at end to a value suitable for continuing
   decryption. */
```

```
void des_cbc_decrypt(DESContext *ks, unsigned char *iv, unsigned char *dest,
                     const unsigned char *src, unsigned int len);

/* Encrypts in CBC mode using triple-DES. */
void des_3cbc_encrypt(DESContext *ks1, unsigned char *iv1,
                      DESContext *ks2, unsigned char *iv2,
                      DESContext *ks3, unsigned char *iv3,
                      unsigned char *dest, const unsigned char *src,
                      unsigned int len);

/* Decrypts in CBC mode using triple-DES. */
void des_3cbc_decrypt(DESContext *ks1, unsigned char *iv1,
                      DESContext *ks2, unsigned char *iv2,
                      DESContext *ks3, unsigned char *iv3,
                      unsigned char *dest, const unsigned char *src,
                      unsigned int len);

#endif /* DES_H */
```

File Name: getput.c

```
/*

getput.h

Author: Tatu Ylonen <ylo@cs.hut.fi>

Copyright (c) 1995 Tatu Ylonen <ylo@cs.hut.fi>, Espoo, Finland
                   All rights reserved

Created: Wed Jun 28 22:36:30 1995 ylo

Macros for storing and retrieving data in msb first and lsb first order.

*/

/*
 * $Id: getput.h,v 1.1.1.1 1996/02/18 21:38:11 ylo Exp $
 * $Log: getput.h,v $
 * Revision 1.1.1.1  1996/02/18 21:38:11  ylo
 *      Imported ssh-1.2.13.
 *
 * Revision 1.2  1995/07/13  01:24:09  ylo
 *      Removed "Last modified" header.
 *      Added cvs log.
 *
 * $Endlog$
 */

#ifndef GETPUT_H
#define GETPUT_H

/*------------ macros for storing/extracting msb first words -------------*/

#define GET_32BIT(cp) (((unsigned long)(unsigned char)(cp)[0] << 24) | \
                       ((unsigned long)(unsigned char)(cp)[1] << 16) | \
                       ((unsigned long)(unsigned char)(cp)[2] << 8) | \
                       ((unsigned long)(unsigned char)(cp)[3]))

#define GET_16BIT(cp) (((unsigned long)(unsigned char)(cp)[0] << 8) | \
                       ((unsigned long)(unsigned char)(cp)[1]))

#define PUT_32BIT(cp, value) do { \
  (cp)[0] = (value) >> 24; \
  (cp)[1] = (value) >> 16; \
  (cp)[2] = (value) >> 8; \
  (cp)[3] = (value); } while (0)

#define PUT_16BIT(cp, value) do { \
  (cp)[0] = (value) >> 8; \
  (cp)[1] = (value); } while (0)

/*------------ macros for storing/extracting lsb first words -------------*/

#define GET_32BIT_LSB_FIRST(cp) \
  (((unsigned long)(unsigned char)(cp)[0]) | \
  ((unsigned long)(unsigned char)(cp)[1] << 8) | \
```

```
  ((unsigned long)(unsigned char)(cp)[2] << 16) | \
  ((unsigned long)(unsigned char)(cp)[3] << 24))

#define GET_16BIT_LSB_FIRST(cp) \
  (((unsigned long)(unsigned char)(cp)[0]) | \
  ((unsigned long)(unsigned char)(cp)[1] << 8))

#define PUT_32BIT_LSB_FIRST(cp, value) do { \
  (cp)[0] = (value); \
  (cp)[1] = (value) >> 8; \
  (cp)[2] = (value) >> 16; \
  (cp)[3] = (value) >> 24; } while (0)

#define PUT_16BIT_LSB_FIRST(cp, value) do { \
  (cp)[0] = (value); \
  (cp)[1] = (value) >> 8; } while (0)

#endif /* GETPUT_H */
```

File Name: messages.c

```
#include "shared.h"


void SendFatalError(int connectFD)
{
  char buffer[8];
  ssize_t bytesWritten;

  sprintf(buffer, "%c\n", FATAL_ERROR_MSG);
  bytesWritten = WriteMessage(connectFD, buffer, strlen(buffer));
  if(bytesWritten <= 0)
    {
      ERROR("SendFatalError: sending fatal error failed.");
      return;
    }

  DEBUG("SendFatalError: sent fatal error message.");
}


/*
WriteMessage() writes length chars of buffer to the socket connectFD.
WriteMessage returns the number of bytes written.  If there is a fatal
error, it returns -1 instead.
*/
long WriteMessage(int connectFD, char buffer[MAXLEN], unsigned long totalBytes)
{
  ssize_t bytesWritten;
  long currentPtr;

  /* Sanity Check -- we don't want to write too much */
  if(totalBytes > MAXLEN - 1)
    {
      ERROR("WriteMessage(): Message exceeds MAXLEN.");
      return(-1);
    }

  /* Make sure the message is not empty */
  if(totalBytes <= 0)
    {
      ERROR("WriteMessage(): No message to write.");
      return(-1);
    }

  currentPtr = 0;

  for(;;)
    {
      /* Try to write the rest of the message */
      bytesWritten = write(connectFD, &(buffer[currentPtr]),
                           totalBytes - currentPtr);

      /* the write got interrupted, retry */
      if(bytesWritten == -1 && errno == EINTR)
        continue;
```

```
      if(bytesWritten <= 0)
        /* message could not be sent, error! */
        return(-1);

      /* Account for what we did write */
      currentPtr += bytesWritten;

      if(currentPtr > totalBytes)
        ERROR("WriteMessage(): write overrun.");

      /* If we managed to write it all, break out of the loop */
      if(currentPtr >= totalBytes)
        break;
    }

  return(currentPtr);
}


/*
ReadMessage() reads from connectFD, storing everything in buffer until
it gets a '\n', at which point it returns the number of bytes which
have been read.  If the read times out or it reads more than MAXLEN it
returns 0 (meaning the read failed)
*/
long ReadMessage(int connectFD, char buffer[MAXLEN])
{
  long currentPtr;
  ssize_t bytesRead;
  int retries;

  currentPtr = 0;
  retries = 0;

  for(;;)
    {
      /* read in a character */
      bytesRead = read(connectFD, &(buffer[currentPtr]), 1);

      /* the read got interrupted, retry */
      if(bytesRead == -1 && errno == EINTR)
        continue;

      if(bytesRead <= 0)
        if(retries < 10)
          {
            retries++;
            continue;
          }
        else
          /* message was incomplete, error! */
          return(-1);

      /* if this is a line feed, we have the entire message */
      if(buffer[currentPtr] == '\n')
        break;

      /* read in another char */
      currentPtr++;
```

```
      if(currentPtr > MAXLEN)
        ERROR("ReadMessage(): buffer overrun.");

      /* make sure we haven't exceeded MAXLEN
         no more room for a null at the end */
      if(currentPtr >= MAXLEN - 1)
        {
          ERROR("ReadMessage(): Message exceeded MAXLEN.");

          /* message was too long, error! */
          return(-1);
        }
    }

  /* Null terminate the end of the string */
  currentPtr++;
  buffer[currentPtr] = '\0';

  /* return the length of the message */
  return(currentPtr);
}
```

File Name: protocol.h

```
/* Minilpd Protocol Header File.
   Shared by server and mini-supervisor */

/*************************************************************************

   Protocol Description:

   UnEncrypted Job:

   (1) The server sends a START_UNENCRYPTED_JOB_MSG to the mini-supervisor.

   (2) If the mini-supervisor is busy with a job, it replies with a
       BUSY_MSG and closes the connection.  The server waits and then
       goes to (1).

       If the mini-supervisor only accepts encrypted connections, it
       replies with a NO_UNENCRYPTED_JOB_MSG and closes the
       connection.  The server reports an error.

       Else the mini-supervisor replies with a BEGIN_JOB_MSG.  Goto (3).

   (3) The server then sends a JOB_DATA_MSG with the number of bytes N
       to be sent (this number is limited by NUM_DATA_LEN_CHARS, which
       is the number of base 10 chars which are used to represent the
       data), followed by the N bytes of job data.  N is limited by
       MAX_JOB_LEN.  The server also listens for responses from the
       printer (and responds accordingly).  If there are no more job
       data bytes to be sent, the server waits for the final status
       message, sends a END_MSG and closes the connection, else goto
       (4).

       The mini-supervisor also queries the printer for any errors it
       might have (like out of paper) by reading from the parallel
       port.  If there is data to read, the mini-supervisor sends a
       RESPONSE_DATA_MSG with the size in bytes of the response from
       the printer, followed by the response.  The number of bytes in
       a response message is at most MAX_RESPONSE_LEN.

   At any time, either side may respond with a FATAL_ERROR_MSG, in
   which case the connection closes.  The connection also closes if
   either side sends an invalid protocol message (effectively a
   FATAL_ERROR_MSG is just another invalid protocol message).


   Encrypted Job:

   (1) The server sends a START_ENCRYPTED_JOB_MSG to the mini-supervisor.

   (2) If the mini-supervisor is busy with a job, it replies with a
       BUSY_MSG and closes the connection.  The server waits and then
       goes to (1).

       Else the mini-supervisor replies with a SEND_KEY_MSG, followed
       by the current job number.  Goto (3).

   (3) The server responds with a KEY_MSG, a pair (the secret key and
```

```
       job number) encrypted by the mini-supervisor's public key.

   (4) The mini-supervisor decrypts the server's message and compares
       the job number in the pair to the current job number.  If they
       match, the mini-supervisor sends a BEGIN_JOB_MSG.
       Otherwise, it sends a JOB_NUMBER_MISMATCH_MSG and closes the
       connection.

   (5) The server then sends a JOB_DATA_MSG with the number of bytes N
       to be sent (this number is limited by NUM_DATA_LEN_CHARS, which
       is the number of base 10 chars which are used to represent the
       data), followed by the N bytes of job data, encrypted with the
       secret key it sent to the mini-supervisor.  N is limited by
       MAX_JOB_LEN.  The server also listens for responses from the
       printer (and responds accordingly).  If there are no more job
       data bytes to be sent, the server waits for the final status
       message, sends a END_MSG, and closes the connection, else goto
       (4).

       The mini-supervisor also queries the printer for any errors it
       might have (like out of paper) by reading the parallel port.
       If the printer has data, the mini-supervisor sends an encrypted
       RESPONSE_DATA_MSG with the size in bytes of the response from
       the printer, followed by the response.  The number of bytes in
       a response message is at most MAX_RESPONSE_LEN.

   At any time, either side may respond with a FATAL_ERROR_MSG, in
   which case the connection closes.  The connection also closes if
   either side sends an invalid protocol message (effectively a
   FATAL_ERROR_MSG is just another invalid protocol message).

**************************************************************************/

/* MESSAGE MEANING DEFINES */

/*      Message Name                 #           Protocol                    */
#define START_UNENCRYPTED_JOB_MSG   'A'       /* "x\n"                       */
#define START_ENCRYPTED_JOB_MSG     'B'       /* "x\n"                       */
#define NO_UNENCRYPTED_JOB_MSG      'C'       /* "x\n"                       */
#define BUSY_MSG                    'D'       /* "x\n"                       */
#define BEGIN_JOB_MSG               'E'       /* "x\n"                       */
#define SEND_KEY_MSG                'F'       /* "x<job number>\n"           */
#define KEY_MSG                     'G'       /* "x<key data length>\n"      */
#define JOB_NUMBER_MISMATCH_MSG     'H'       /* "x\n"                       */
#define FATAL_ERROR_MSG             'I'       /* "x\n"                       */


#define MAXLEN              1024               /* max # bytes of any message  */
#define MINIPORT            4000
#define MAX_JOB_LEN         8192
#define MAX_RESPONSE_LEN    1024
#define MAX_KEY_LEN         (sizeof(des_cblock) * 6 + 12)
                            /* 4 keys + jobNum + padding */

typedef unsigned int        FourBytes;
```

File Name: shared.h

```
#include "reporting.h"
#include "protocol.h"
#include "des.h"

#include <errno.h>
#include <syslog.h>
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <sys/uio.h>
#include <unistd.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <assert.h>


/*
#include <sys/param.h>
#include <sys/file.h>
#include <sys/stat.h>
#include <sys/un.h>
#include <pwd.h>
#include <termios.h>
*/


/* DEFINES */
#define FATAL         -1
#define NO_ERR         0
#define GOT_EOF        1
#define FULL           2

/* Key Structure */
typedef unsigned char des_cblock[8];

typedef struct _keys
{
  DESContext       key1;
  DESContext       key2;
  DESContext       key3;
  des_cblock       ivec1;
  des_cblock       ivec2;
  des_cblock       ivec3;
} Keys;

/* PROTOTYPES */
/* data.c */
int GetData(int getFromFD, char *dataBuffer, long *dataBytes, long bufferBytes);
int SendData(int sendToFD, char *dataBuffer, long *dataBytes, long bufferBytes);
int EGetData(int getFromFD, char *dataBuffer, long *dataBytes,
```

```
             long bufferBytes, Keys *getKeys);
int ESendData(int sendToFD, char *dataBuffer, long *dataBytes,
              long bufferBytes, Keys *sendKeys);

/* messages.c */
void SendFatalError(int connectFD);
long WriteMessage(int connectFD, char buffer[MAXLEN], unsigned long totalBytes);
long ReadMessage(int connectFD, char buffer[MAXLEN]);

/* utils.c */
int maximum(int i, int j);
int GetMasterKeys(Keys *masterKeys);
void ZeroKeys(Keys theKeys);
```

File Name: utils.c

```
#include "shared.h"

int maximum(int i, int j)
{
  if(i > j)
    return i;
  else
    return j;
}


int GetMasterKeys(Keys *masterKeys)
{
  unsigned char  mkey1[8]={0xf0,0xe1,0xd2,0xc3,0xb4,0xa5,0x96,0x87};
  unsigned char  mkey2[8]={0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef};
  unsigned char  mkey3[8]={0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10};

  des_set_key(mkey1, &masterKeys->key1);
  des_set_key(mkey2, &masterKeys->key2);
  des_set_key(mkey3, &masterKeys->key3);

  memset(masterKeys->ivec1, 0, sizeof(des_cblock));
  memset(masterKeys->ivec2, 0, sizeof(des_cblock));
  memset(masterKeys->ivec3, 0, sizeof(des_cblock));

  return(NO_ERR);
}


void ZeroKeys(Keys theKeys)
{

  return;
}
```