Feasibility of GNU/Linux as the OS for a PC-based Medical Product

by

Steven B. Lustbader

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

December 20, 2002

Copyright 2002 Steven B. Lustbader. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and 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
December 20, 2002

Certified by

Bob Desmarais
VI-A Company Thesis Supervisor

Certified by

Barbara Liskov
M.I.T. Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Theses
Feasibility of GNU/Linux as the OS for a PC-based Medical Product

by

Steven B. Lustbader

Submitted to the
Department of Electrical Engineering and Computer Science

December 20, 2002

In Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Linux has become a viable alternative to Windows in recent years. This investigation looks at the feasibility of porting the software for a PC-based medical device to Linux. Using an open-source operating system frees developers from the constraints imposed by relying on a single company for the development platform. Several porting methods are considered. The port method chosen allows development on the Windows version to continue while simultaneously testing on Linux, without creating separate versions of the software. Differences in the way the software interacts with the operating system and with the hardware have to be addressed. A Linux environment was created in which to run the software and determine how to reconcile these differences. No major hurdles to using Linux exist, so it appears to be a viable platform on which to conduct future development.

Thesis Supervisor: Barbara Liskov
Title: Ford Professor of Engineering, Associate Department Head, Computer Science & Engineering
1.0 Introduction

In the 10-plus years of the development of the GNU/Linux operating system, it has been transformed from a fringe OS used by a small group of programmers to a mainstream system with a rapidly growing share of the operating system market. With this wider user-base, it is now feasible to base commercial systems on the operating system, and, in some cases, it is beneficial to use Linux over the more popular Windows operating systems. This thesis will examine the suitability of basing a PC-based medical product, referred to here as Orion, on Linux and the efforts required to port Orion to Linux from Windows.

Windows XP Embedded (XPE) provides all the necessary kernel services that Orion needs. The video game industry drives Microsoft’s DirectX towards faster and more powerful graphics, a trend Orion makes use of. There are many programmers who are familiar with the development tools used in Windows programming, commonly Microsoft’s Visual Studio, so finding staff is easy and little time is lost in familiarizing newly hired employees to the development environment. In addition, Microsoft provides a great deal of support to Windows developers through the Microsoft Developers’ Network (MSDN).

These advantages held by Windows come at a price. Being dependent on only one company limits one’s flexibility and range of options should that company decide to head in a new direction. Orion depends on Microsoft for audio, video, windowing, authentication, localization, etc. This reliance makes it easier for Orion’s developers, but it can leave them with a large problem if Microsoft decides to focus its efforts in a new way. Microsoft already uses strict licenses to control how its technologies, such as
DirectX, can be used with third-party software, preventing any open-source code from being used in some situations. This restriction can limit development, and Microsoft has the ability to add more restrictions if it wants. In addition, each product shipped requires a Windows license, and the proprietary nature of the operating system can limit customization for specific needs. Windows XPE is more customizable than normal XP, but it still has restrictions on what can be done.

Linux is a possible alternative to Windows for Orion. Linux is very cheap; the operating system can be obtained through either a free download or for a small one-time fee to purchase it on CD. There are no licensing fees and therefore no fees that must be paid based on the number of products sold. In addition, Linux is open-source, which allows easy changes to the base operating system. These changes can be simple bug fixes or custom kernel modifications that augment or restrict the functionality of the operating system as needed.

Of course, Linux has its own disadvantages. Its open-source nature has its own legal issues concerning how it can be used commercially, and while kernel changes are easy, they would have to be done in-house, needing additional developers. Since Windows is a more popular operating system than Linux, there are more developers who are familiar with software development on Windows than on Linux. Another disadvantage is the lack of a central authority for Linux, including no equivalent to the MSDN for Linux development.

The goal of this thesis is to determine the issues involved with porting Orion to Linux. Ideally, the source code will remain mostly unchanged from the Windows source.
Having Orion working on the two operating systems provides increased flexibility during the porting process before moving completely to Linux.

The rest of this thesis is organized as follows. The next section describes related work. Section 3.0 continues by describing the architecture of Orion, and Section 4.0 discusses the possible methods of implementation and the pros and cons of each. Following that, Section 5.0 explains how the actual port was done, and Section 6.0 describes future projects that may be done to continue the work started here. Section 7.0 closes with a discussion.

2.0 Related Work

Very little has been published about porting applications from Windows to Linux. This lack of publications may be because porting small applications does not seem worthy of publication and publishing papers on porting large systems runs into the problem of revealing company secrets. Corel falls into this latter category; they ported their WordPerfect office suite to Linux, but they have not released their source code or described in detail the process they used [1]. As such, no papers similar to this one have been found.

There are several how-to and frequently-asked-question (FAQ) documents published on the Internet pertaining to porting applications to Linux, but most involve large changes to the source code. Markus Neifer at IBM, for example, has written about converting a program written using the Microsoft Foundation Classes (MFC) for Windows to wxWindows [2]. WxWindows is an open-source graphical user interface (GUI) toolkit that works on both Windows and Linux [3]. This was precisely the method
mentioned in [1] that they wanted to avoid, since it would require a major rewrite of the source code.

Other articles discuss porting software at a very high level. In particular, Geoffrey Noer raises several questions that need to be answered before even starting a port, such as emulation versus native compilation [4]. It also recommends several tools depending on the porting method chosen, but offers no help or guidance beyond that.

3.0 System Architecture

As seen in Figure 1, Orion is actually made up of three separate “stations.” Station zero is a Windows PC, connected through a communications subsystem to stations one and two, which are two pieces of custom hardware, each running an embedded operating system. Station zero is also connected to several peripherals via a USB interface.

![Figure 1 – The overall architecture of Orion](image)

Station zero is made up of many sub-applications. Each sub-application is encapsulated in the form of a dynamic link library (DLL), all of which are loaded by an executable.

For development purposes, it is possible to run Orion without the hardware for stations one and two. There are two Windows applications that simulate the behavior of
those stations, and the communications subsystem is slightly modified to accommodate the simulators. This Windows-only simulation environment is what is being ported to Linux. The issues involved in implementing the Linux version of the communications driver for the target hardware are not addressed in this paper.

4.0 Possible Methods of Implementation

The easiest method is to run Orion on top of a Windows emulator, such as VMWare [5]. This software provides virtual hardware on top of Linux, allowing Windows to be installed to run normal Windows binaries. This simplicity, however, comes at the expense of speed; the extra emulation layer decreases the efficiency of system calls. Orion must be responsive: speed and efficiency are vital, so emulation is not a preferred option. In addition, VMWare may have problems with Orion’s custom hardware and peripherals, and this approach would be expensive, requiring licenses for both Windows and VMWare.

Another method is a large-scale rewrite of the source code to use native Linux libraries. Most of the work here is in the user interface, threading, and localization/internationalization, a large part of Orion’s PC software. There are also smaller, minor differences between Linux’s GNU C++ compiler and Microsoft’s Visual C++ compiler that would need to be addressed. The first problem with this method is that Orion has been in development for three years; the effort to implement the OS and GUI parts would be substantial. Second, the result would be two separate versions of Orion: a Windows version and a Linux version. With this separation, the porting process would be slowed, as new features would need to be implemented twice. Eventually
Orion would only run on Linux, but until that time, it must run on both operating systems.

A third alternative would be a cross-platform toolkit, which would allow development from a unified source base that would run on both platforms. Sun’s Java [6] is an example. Rewriting Orion in Java would allow it to run on both Linux and Windows from one source base, so porting could be done while adding new features to the Windows product. This would require rewriting all of the PC source code, although the C++ to Java conversion is not as major as converting between less related programming languages. Other options of this type are wxWindows and the Qt toolkit [7], a cross-platform C++ GUI toolkit. Both of these options would provide standard user-interface code for both platforms, although threading and internationalization issues would still remain.

The last alternative that was considered for porting Orion to Linux is to implement the Windows libraries directly on Linux. This method would allow the current source code to be used, without the performance decrease associated with emulators. The Wine project [8] is an open-source effort to provide such an implementation. Wine is made up of two parts, a binary loader to load unmodified Windows executables, and WineLib, a Linux implementation of the Windows libraries. WineLib provides the header files and linker settings necessary to compile Windows applications on Linux. Wine is not a traditional emulator with the usual performance decrease. It does not emulate hardware; it allows Windows executables and DLLs to call other Windows DLLs the same way they would on Windows. Orion can use Wine with WineLib by natively compiling some of Orion’s libraries using WineLib and then
running the program with Wine. Orion can call the natively compiled libraries as if they were normal Windows libraries. This method is the way Corel chose to port WordPerfect to Linux.

Using Wine and WineLib, it should be possible to run Orion on Linux with only relatively minor changes to the source code. Unfortunately, Wine is still in the alpha stage of development, so it is almost constantly updated with bug fixes and new features. In addition, without access to the source code of Windows, most of the implementation is based on Microsoft’s published documentation and trial-and-error. As such, Wine is playing catch-up with Microsoft; as Windows gains new features, those features have to be implemented in Wine. Since Wine is not a commercial project, implementing new features is not always given a top priority by Wine’s many developers.

Table 1 below shows the various porting methods and their relative advantages and disadvantages. Port Speed is an indication of how long it will take to perform the actual port from Windows to Linux, and Execution Speed is an indication of how fast Orion will run on Linux after it has been ported. Source Code Unity refers to how similar the Linux and Windows source code are; source code that is completely the same for both platforms (such as if Java were used) received a rating of “+++,” while source code that is completely separate is rated “+.” Methods requiring minor changes to the source (due to compiler differences, for example) received a rating of “++.” For this project, port speed and source code unity are the most important aspect, as long as execution speed is adequate. Based on these criteria, it is apparent from the table that the Wine/WineLib combination is the best of the options discussed here.
Table 1 – Comparison of different porting methods (+: bad, ++: OK, +++: good)

<table>
<thead>
<tr>
<th>Method</th>
<th>Port Speed</th>
<th>Execution Speed</th>
<th>Source Code Unity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulation</td>
<td>+++</td>
<td>+</td>
<td>+++</td>
</tr>
<tr>
<td>Complete Linux rewrite</td>
<td>+</td>
<td>+++</td>
<td>+</td>
</tr>
<tr>
<td>Java</td>
<td>+</td>
<td>+</td>
<td>+++</td>
</tr>
<tr>
<td>Cross-platform GUI toolkit</td>
<td>+</td>
<td>+++</td>
<td>GUI – +++</td>
</tr>
<tr>
<td>Re-implement Windows Libraries</td>
<td>+</td>
<td>+++</td>
<td>Everything Else – +</td>
</tr>
<tr>
<td>Wine/WineLib</td>
<td>++</td>
<td>+++</td>
<td>+</td>
</tr>
</tbody>
</table>

5.0 Design and Implementation

With Wine chosen as the tool used to perform the port, it is possible to run the Orion executable and the two simulator executables almost unchanged. The areas that Wine needs to handle are the Orion-to-operating-system interactions. These interactions can be grouped into major categories: graphical user interface, threads, internationalization, hardware interaction, and file I/O. The GUI category can be further broken down into APIs: MFC, ATL, and straight Win32. Most hardware interaction, including graphics and sound, is handled through DirectX. Figure 2 shows the interactions between Orion and the operating system (the “deltas” are explained in section 5.3). Under Windows, Orion makes systems calls using MFC, ATL, etc., that are handled directly by the operating system. Under Linux, however, another layer is required to convert the Windows system calls to Linux system calls. Wine provides this layer, allowing Orion to continue to call the original Windows functions, which actually call native Linux functions.
The first step is implementing the communications subsystem in Linux. Unlike the rest of Orion, the communications subsystem is a kernel-mode driver. Wine can only run user-space Windows applications, so Wine cannot run this subsystem.

The peripherals do not provide important functionality, so the core of Orion can still be run without them. As such, USB these drivers did not need to be ported for this investigation.

The next step is to run a very minimal set of sub-applications with Wine. This step tested both the implementation of the communications subsystem and the capabilities of Wine as they apply to Orion. These sub-applications are the original unmodified Windows executable and dynamic link libraries. Only one library that accesses the communications subsystem needed modification. The components chosen for this step make up the infrastructure of Orion.
After a minimal Orion system is run successfully with Wine, other sub-applications were added. More issues were discovered at this point, both with the communications subsystem and with Wine.

### 5.1 Communications Subsystem Implementation

The communications subsystem is fundamental to Orion, so it is a logical place to begin. The Windows version of Orion uses a kernel-mode driver that has two modes, normal and simulation, with a communications interface to access the driver from inside the application. The normal mode is a standard driver that provides access to the hardware. The simulation mode is used by developers who do not have the actual hardware installed in their computers. In simulation mode, the subsystem passes information between the simulation stations using shared memory. The simulator architecture is illustrated in Figure 3.

![Architecture of the communications subsystem simulator in Windows](image)

**Figure 3 – Architecture of the communications subsystem simulator in Windows**

The two modes of the communications subsystem share a great deal of code, so they were both done in kernel space, with the user-space communications interface to access the driver. This choice was made more out of convenience than out of necessity;
while simulating the hardware, the subsystem does not need to run in kernel space, since there is no hardware for it to access.

The communications subsystem is implemented as a normal Linux application for this investigation, rather than as a kernel-mode driver. The stations use inter-process communication (IPC) to pass messages to each other through the subsystem, as shown in Figure 4.

For simplicity’s sake, pipes were chosen as the IPC mechanism with which to communicate with the subsystem. The code originally in place that used shared memory would have had to be completely rewritten. It was faster to implement the communications subsystem using pipes than to reimplement it with shared memory.

![Figure 4 – Architecture of the communications subsystem simulator in Linux](image)

With most of Orion remaining a Windows application and the communications subsystem now a Linux application, some sort of layer needs to be inserted to allow the two parts to talk. WineLib allows for either Windows applications to make Linux system calls or Linux applications to make Windows calls. To keep the communications subsystem simpler, it was implemented as a pure Linux application without using
WineLib. Therefore, the Windows side of the subsystem needs to be able to make Linux calls. An additional DLL (the WineLib Wrapper layer in Figure 4) was created to provide this interface. When compiling the Windows part of the communications interface, this DLL was filled with stub functions and used only for linking with the proper symbol names. On the Linux side, a WineLib wrapper DLL was created that actually called the appropriate Linux functions, and at run time, Wine linked Orion with that DLL rather than the stub used in Windows. This approach made it very easy to call most Linux functions from inside Windows applications without recompiling them using WineLib; only a small wrapper library must be compiled with WineLib.

5.2 Minimal Orion System

With the communications subsystem in place, it became possible to run a subset of the Orion applications and the station simulators in Linux. This subset makes up the infrastructure of Orion, providing basic services to the rest of the system. These services include starting the three stations and the communication subsystem, initializing communications between the stations, handling errors, displaying the user interface, and accessing sound and peripherals.

At this point, several problems were encountered with Wine. These problems were mostly hardware interaction issues that can be attributed to Wine being alpha software. Wine’s current Windows compatibility target is Windows 98, so support for later operating systems, such as Windows 2000 and Windows XP, is not complete. As a result, many functions implemented in those later versions have not been implemented in Wine yet, and in many cases there were not even stubs for them. Fortunately, most of these unimplemented functions were not needed for the basic functionality desired in this
proof-of-concept, and minimal function stubs were all that was required to solve this problem.

In addition, Wine’s DirectX support is not fully up-to-date, so there were sound and video problems. Sound was deemed to be nonessential, so no time was wasted in trying to update Wine’s DirectSound support, but video is very important. For a portion of the display, Orion uses DirectDraw. Wine only has DirectDraw implemented for 16-bit color, while Orion requires 32-bit color. This difference does not cause problems, since Orion detects it and does not try to use DirectDraw. Therefore, that section of the screen remains blank. The information needed to draw that section is correct, however, and when Wine adds 32-bit color support to DirectDraw, this problem will disappear.

Other minor issues also existed, such as Windows API calls that were implemented in Wine for the most common usages but not for less common ways of calling them. As an example, most times PostMessage is called, it is passed a handle to the window to which the message is being sent. Orion, however, at times calls PostMessage with a NULL window handle, which is allowed in the specification of the function, although it is not commonly used this way. Wine did not handle this case, since no application previously run with Wine had called PostMessage that way. The solutions for this instance and for other similar problems were minor, simply bringing the implementation up to date with Microsoft’s specifications.

Other issues were related to differences between Linux and Windows. Most significantly, these differences included the ways basic hardware, such as floppy drives and serial ports, are accessed. For example, in Windows, the floppy drive is always available, regardless of whether or not there is a disk in the drive. In Linux, however, the
drive must have a disk in it and be mounted to be accessible. This difference led Orion to believe the computer had no floppy drive when run in Wine. Like before, this issue and other similar ones did not affect Orion’s basic functionality, so no time was wasted reconciling these differences.

With the Wine and Linux problems either fixed or circumvented, the minimal Orion was successfully loaded.

5.3 Complete Orion System

Not surprisingly, when including all the applications, more problems were encountered. Several of the applications used functions in Windows DLLs not included with Wine. Fortunately, Wine has the ability to use both its own, natively compiled libraries or libraries copied from a Windows installation. Copying the needed libraries to Linux solved the problem. This solution, however, has its own problem, in that it is unclear what the license on many DLLs is. Some are redistributable and some are not, and different versions of the same DLL may have different restrictions. These legal issues may prevent use of the DLLs in a commercial, non-Windows environment, but that is unclear. Fortunately, the use of the Microsoft DLLs is only temporary, because at least two of the four DLLs that were copied are currently being added to Wine. Once they are in place, the license problems will not be an issue.

A new problem that appeared in this stage involved the way some Windows API calls were made. In a few locations, the developer of that application made assumptions about what the call would return, sometimes assuming that the call would always succeed. In Windows, these assumptions were acceptable, even though they may not have followed the specification of the function, since the runtime environment of Orion
was understood. With Wine, however, these assumptions led to crashes. For example, at one point, an application attempts to call GetModuleBaseName, which should return a string with the base name of the running module (library or executable). This string is then parsed for a certain substring. The application assumed GetModuleBaseName would always succeed, so no error checking was performed on the returned value. In Wine, however, that function is not fully implemented. Therefore, the string it returned was invalid, and without error checking, parsing of that string caused a crash. Although this portion of code worked in Windows, it was not completely correct, and Wine revealed that shortcoming. Simply adding error checking in that instance and other similar ones solved the problem. These minor changes, which do not affect functionality in Windows but help in Linux, are represented by the “deltas” in Figure 2.

A slightly more serious problem occurred upon startup when loading all the applications. One of the applications simply crashed right away, before the system was even fully initialized. This problem was traced back to a problem with Wine’s implementation of CreateWindow. CreateWindow is a complex function tied deeply into Wine’s infrastructure, so debugging it was extremely difficult. The application in which the crash occurred was not vital, nor did any other application depend on it, so it was possible to remove it. This solution circumvented the problem with very little loss of functionality. Unfortunately, the Orion setup that was being run was no longer complete, although it was very close.

5.4 Current Status

Overall, a majority of Orion can be run in Linux. As mentioned previously, one application was removed because it crashed on startup. Another application was
removed because its sole purpose was to control a USB peripheral; since that peripheral was not there, and the application incessantly complained about not being able to access it, that application was also removed, with no extra loss of functionality. All other applications run successfully. There is no sound, nor is there video in the section of the display that uses DirectDraw.

6.0 Future Work

The next step in porting Orion to Linux is enabling the full display. The DirectDraw portion of the screen is very important for the full use of Orion, even while simulating. Wine needs to be augmented to include support for DirectDraw in color depths other than 16 bits-per-pixel.

Enabling full hardware support, which was skipped during this investigation, also needs to be done. Most importantly, the communications subsystem must be implemented so Linux can be used with production hardware. Drivers for the USB peripherals also have to be written, and the other minor differences in hardware access between Linux and Windows need to be reconciled.

Another area that was left unexplored was Orion’s ability to hide Linux. As an embedded application, especially a regulated medical device, it is important that the user not be able to access the operating system behind the software. It could cause problems if the user were able to change settings outside of the software in an unregulated way. Windows XPE does provide some capabilities to lock down operating system functionality behind Orion but not as much is possible with access to kernel source code. It may be possible to remove more unneeded or unwanted functionality in Linux, since
the kernel source code is accessible. This ability would make Orion more secure, both from outside threats and from the user.

The applications that were left out of the “complete” system need to be included. After the USB peripherals are enabled, one of these can be added. The other skipped application requires fixes to Wine before it can be included. In addition, there is separate reporting software used by Orion that was not tested with Wine.

Another future project is natively compiling some applications, either for efficiency or to make use of some Linux feature. These components would be part of the “deltas” in Figure 2. Exactly which applications could benefit from this recompilation would need to be determined. Compiling an application natively, even without making use of the Linux API, may require minor change due to compiler differences. In addition, WineLib’s support for ATL and MFC is minimal, since those APIs are very large and complicated. Therefore, natively compiling an application that uses either API would require separating it into two DLLs. One DLL would contain the user interface code that uses ATL or MFC and would remain compiled in Windows. The UI DLL would then call another DLL that provides the backend functionality, which would be compiled in Linux using WineLib.

7.0 Conclusion

This investigation revealed no reason why Linux could not be used for Orion. Although several problems were encountered, they all appear to be resolvable. Wine has proven to be a good platform on which to perform the port. It allows Orion’s developers to continue to add features to the Windows version without the need to reimplement those
features for Linux. It also runs Orion with no noticeable loss of speed; no benchmarks have been run, but no performance decrease has been visible.

To continue down this path, Orion needs a group of developers to work closely with the Wine and Linux development world. Although most bugs in Wine were fixed quickly, either by the author or by other Wine developers, larger problems require dedicated Wine developers. These problems may include improving DirectX support and fixing Windows XP compatibility. In addition, developers familiar with the Linux kernel are needed to make any needed modifications to the kernel.

Linux does have some drawbacks. Orion will lose some of the benefits it gained by riding trends in the industry. Video games have driven video hardware and related software technologies to high performance levels, so little effort was needed to provide Orion with excellent video performance. This push by the video game industry has benefited Windows more than Linux, although Linux is starting to catch up. It may be some time before Linux has reached the level of maturity that Windows has reached in this area. Any other areas in which Microsoft has become a driving force will also lag behind in Linux, but such gaps may not affect Orion.

Since Wine has somewhat separated the development of Orion from the operating system it runs on, investigation of Linux’s suitability can continue without affecting current development of Orion. Orion now can be run on both Windows and Linux temporarily, and eventually it can be moved completely to Linux.

8.0 References


