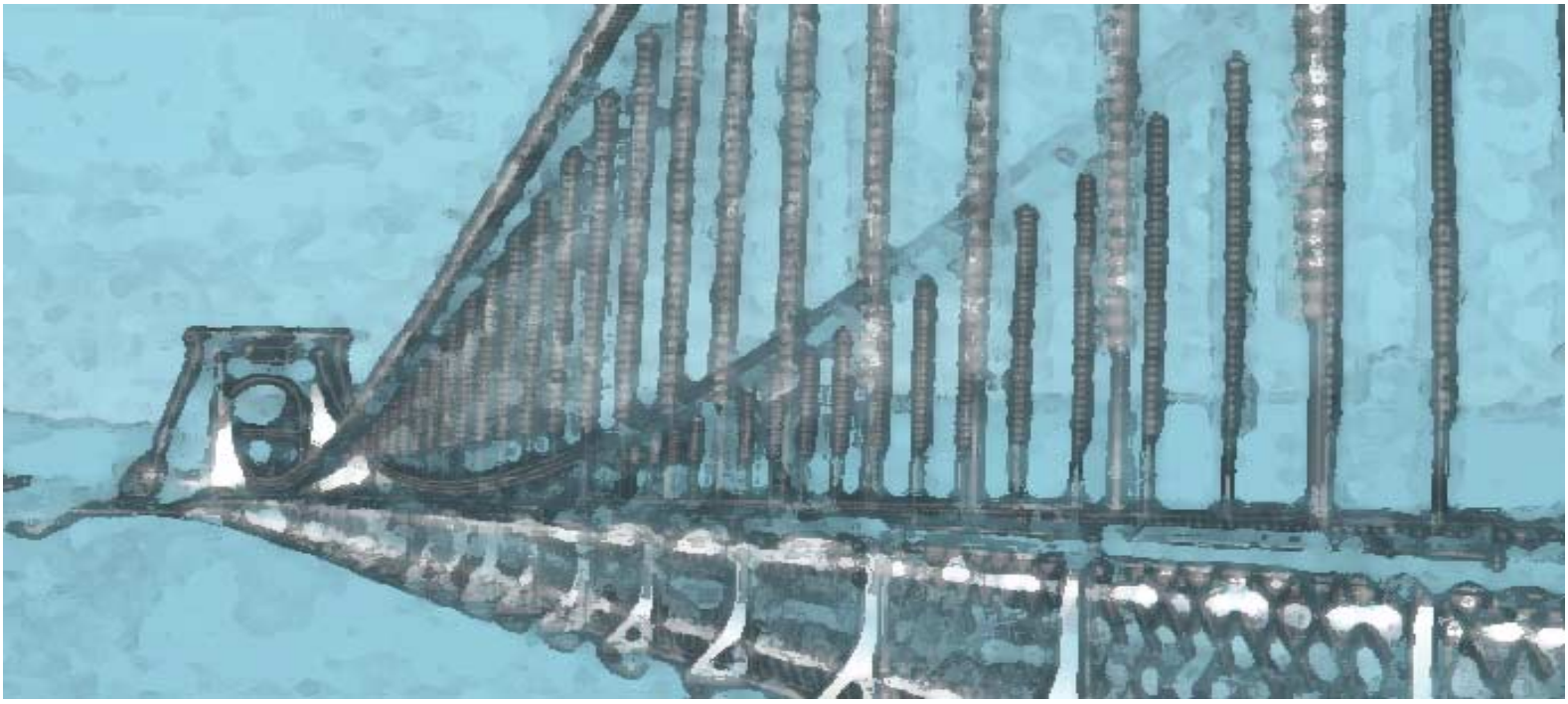
Porting from UNIX to Linux

Issues and Advice for a Successful X/Motif Migration



Integrated Computer Solutions Incorporated

The User Interface Company™

201 Broadway
Cambridge, MA 02139
617.621.0060
info@ics.com

www.ics.com

Porting from UNIX to Linux:

Issues and Advice for a Successful X/Motif Migration

Table of Contents

Porting X/Motif Applications to Linux

Introduction

Many organizations are using or examining Linux as a lower cost development and deployment platform for a wide range of applications. In the web server domain, Apache and Linux have proven themselves in thousands of installations. In many cases, organizations have had an easy job of deploying web applications for that environment. Their job is simplified because the applications were developed from the ground up on Linux.

Companies with older X/Motif applications are faced with a different problem. They often have code which may have been developed 5 to 10 years ago. It probably runs on hardware which is now costly to maintain. The question they are facing is whether they can realize a savings by porting their Motif applications to Linux.

To help answer that question, this paper examines some of the issues you are likely to encounter when porting your X/Motif applications from another Unix platform to Linux. In particular we'll look at

- Making an inventory
- Picking the right Motif version
- Compiler issues
- Hardware issues
- X server differences
- Operating System differences

We'll also discuss strategies you can use to help estimate these costs before committing to a port and provide specific advice about how to minimize risks.

Making an inventory of your application

The most important consideration before starting a port is to answer the question "Do I have all the components I need?". To answer that, you'll have to examine your code base and identify

- all the source files
- any special tools other than the compiler to build it
- any internally built custom Widgets
- any 3rd party custom Widgets
- any other 3rd party libraries used
- any database APIs used

Once this is known, you'll have divided the application into parts that you have to port, and parts that 3rd parties have ported for you. The critical thing to do now is make sure

that all of the 3rd party components are available for Linux. If not, you'll have to consider the cost of replacing them.

Picking the right Motif

There are two major choices for getting Motif on Linux

- use the one that comes with your Linux distribution;
- get the free OpenMotif distribution from MotifZone.net

We strongly recommend that you develop with a copy of OpenMotif 2.2. Further, we suggest you obtain your own copy rather than using a package that comes with your Linux distribution. By doing this you insulate yourself from changes which you Linux vendor may introduce. You can get either a pre-built binary from MotifZone.org, or get the sources from The Open Group and build it yourself. Even if you opt for a binary version, it is prudent to have the source at hand, both for study and as self-escrow against possible Motif changes in the future.

Changes in Motif from 1.2 to 2.x

For the most part, you will not have to worry about changes from Motif 1.2 to 2.x. The old APIs remained intact, but new widgets and methods were added. If, however, you built your own custom widgets, you will have to upgrade them. There were extensive internal API changes to Motif, so older widget code will not work without some change.

Our recommendation is to upgrade your custom Motif 1.2 based widgets. If you identified a large number of them and you do not have the resources to update them, you might consider remaining at Motif 1.2. For individuals desiring to pursue this route, ICS is able to provide a supported Motif 1.2 for Linux. Of course, if you are also using other 3rd party widgets, they may not be available for Motif 1.2 on Linux anyway, so you will still need to upgrade. In that case, you may find it economical to farm out the sub-project of updating your old widgets to a firm which specializes in X & Motif (allow me to suggest ICS?).

Moving to GCC

You will, very likely, be moving from your existing hardware vendor's C/C++ compiler to GCC, which is the GNU C/C++ compiler. It is provided with all major Linux distributions. There is no reason to buy a proprietary compiler and it will probably make your job more difficult if you do.

Unless all of your code is very new, and especially if you are porting older C code, you will probably need to update function prototypes to ANSI standard. You may also have to clean up a few instances where non-standard code was allowed by the older compiler. There may be a lot of little changes, but none of them should be conceptually difficult.

The good news is that you can estimate this task and even accomplish most of it without purchasing any new software or equipment.

- Get a copy of GCC (version 3.2, but we'll talk about that later) for your current hardware platform. It's almost certainly available. Check the GNU web site for details.
- Try to compile your application with GCC on your current hardware.
- Fix prototypes and #includes and cross check against your old compiler.

In all but the rarest cases you should be able to create a source tree which compiles under both GCC and your hardware vendor's compiler. The application should still run on the older platform and, in most instances, a good deal of it should work right on Linux as well.

Picking the right GCC version

As of November 2003, there are 4 major versions you might find in use - 2.95, 2.96, 3.2.x and 3.3.x. Our advice at this time is to use 3.2.x. More importantly, use a Linux distribution that ships with 3.2.x so that you don't have to fight the crowd. Most third party libraries have the most support for 3.2, and treat 2.95 and 2.96 as an afterthoughts. Many developers have not adopted 3.3 yet.

We cannot emphasize how important it is to stick with the mainstream. Do not use old versions of the Linux kernel. Don't use old GCC (2.95). On the other hand, do not keep installing the newest releases of GCC, because you'll find that 3rd party libraries may not work with it yet. There is an implicit assumption that the GCC shipped with a Linux distribution is the same one used to build the kernel and all the system libraries. If you start to violate that assumption, you end up in pain.

More information

GNU Press publishes Using GCC, which is a contains a wealth of information about the compiler.

Hardware issues

Happily, most C code compiles and runs just fine on a wide variety of hardware. Unfortunately, a large number of interesting programs work with binary data and/or network connections at some point. If your software does that, you will almost certainly have to inspect it for things that might break moving to new hardware. The two major issues are changing byte order (endian-ness) and changing alignment rules.

Byte Order

This table will help you determine the byte ordering that you are porting from and to.

Architecture	Byte order
DEC Alpha	little endian
IBM zSeries	big endian
MIPS (SGI)	big endian
MIPS (DEC DECStation)	little endian
PA-RISC	big endian
PowerPC	big endian
SPARC	big endian
x86	little endian

Little endian architectures store the least significant byte of data in the lowest memory address. Big-endian architectures store the most significant byte first. (There are variants on that, which come up mainly on the DEC VAX, but that is beyond the scope of this paper). For example, if we had the 32-bit integer 0x01020304, it would be laid out in successive memory locations like this

00000004	00000003	00000002	00000001

on a little-endian machine, while on a big-endian machine it would be laid out like this

00000001	00000002	00000003	00000004

If you are like most people reading this paper, you will be moving from SPARC, PA-RISC or PowerPC to x86, so the endian concerns are important. The following example shows what can go wrong.

```
struct timestamp {
            short   year;
            char    month;
            char    day;
            char    hour;
            char    minute;
            char    second;
            short   milliseconds;
            long    data;
} ts;

      . . .
/* load the timestamp from the log file */
read(dataFile, &ts, sizeof(ts));
```

This code may have been working for years in your organization, but it should have been replaced a long time ago. Let's say we had a time stamp of 1-1-2001, 11:22:33.44 and wrote the data file on a SPARC. When we read it back in on an x86, we get 1-1-53511, 11:22:33.11264.

You'll have to carefully examine your code for all the places where you do IO on binary structures and fix them accordingly.

Structure Alignment

A related way things can go wrong is if the compilers on the old and new platforms align elements in a structure differently. In the above example, the *data* element is commonly aligned on the 12th byte of the structure, so that it takes 16 bytes total. It is possible, however, on some platforms with certain sets of compiler switches to end up with *data* aligned on the 10th byte.

The most common case where this might be an issue is when marshalling data for binary IO. Luckily, you'll probably catch instances of this when you look for byte order issues. Another interesting case is when you are writing code which can access the elements of a structure through some table driven means. The way widget resources are defined in the X Toolkit is a great example. Each resource is defined by name, data type and offset from the beginning of the structure (among other things). Widgets do this in a portable way by using the XtOffset macro, which hides this ugly pointer arithmetic

```
(((char*)(&(((struct timestamp *)NULL)->data))) - ((char *) NULL));
```

When you see code which takes a pointer, casts it to a *char **, adds a **constant**, then casts it to something else, you should check that carefully. The constant, in particular, should be treated as guilty until proven innocent, and really should be replaced with a descriptive name and/or a portable macro in any case.

Moving from 32 bits to 64 bits (or visa versa)

It is possible you may be changing the number of bits as well. All we can say is that this will magnify your opportunity to both spot and have problems related to byte order and alignment. On the other hand, X and Motif were 64-bit safe many years ago, so there is a lot of experience about the issues. A web search will turn up lots of hints and tips.

X Server Differences

If your application is mostly forms based and does not try to do anything tricky with managing a lot of colors, you will probably have no problems with X server changes. On the other hand, if you managed an extensive color palette, you may have some work ahead.

Video Hardware Got Better

Most video hardware from 8 to 10 years ago used an 8 bit PseudoColor display. A pixel value from 0 to 255 was used as an index into a hardware colormap, which contained the parameters for the actual color to display. The X server would swap colormaps for windows requiring different colormaps. There was usually a limit to the number of colormaps, so that, once that was reached, windows using a swapped-out colormap, would dramatically change to a palette of inappropriate colors. Much coding effort was likely expended trying to prevent your applications from being badly behaved this way.

Today, the XFree86 X server on Linux supports a limited use of PseudoColor because most current video hardware supports both 16 and 24 bit TrueColor among other display types. Applications do not have to worry about using only 256 colors.

So why is this a problem? Some applications would install their own colormaps, and pre-load them with specific colors at each position. This way, the developer knew, for example, that black would be 0, white would be 1 and red would be 2. They could set colors by using specific indices. It was very efficient, but does not work with more powerful display types.

As we said above, most applications will not encounter this issue. The few that use lots of color, and were built for hardware with limited capabilities will probably need work to correct assumptions that are no longer true.

OS differences

For the most part, the differences between Linux and other Unixes are minimal. Even better, they are often differences which the compiler will detect because APIs are slightly different. It is often just a matter of diligent work to detect and fix all the OS specific differences. Some things you might encounter are:

Multi-threading

Linux uses the POSIX threading API. Solaris provides both POSIX and a Solaris-specific API. If your application used a vendor-proprietary API, you will need to move to the POSIX API. Since the models are similar, this should be very easy. It is also something you can usually attempt on the old platform, before doing the port.

Deprecated library functions

Some library functions are unsafe or deprecated and the GNU compiler will often warn you about them. In particular applications that create temporary files (and use the functions mktemp, tempnam or tmpnam) should use this opportunity to update to the POSIX mkstemp function.

Linux uses bash as the shell rather than ksh

This is usually not a big concern unless you have a large number of shell scripts. Although the application itself may not use the shell, very often your build and test environment might have a lot of scripts.

Further reading

This paper tries to discuss the problems you may encounter from a very high level view. Your specific case will always be just that - specific. You will know if you're porting from HP-UX 10 on PA-RISC. My advice is to search your existing hardware vendors' website for their porting tips. Many of them have a Linux-to- Our-OS guide. Just reverse their instructions for a lot of detailed information.

Finally, every vendor selling Linux is more than happy to provide information about how to port from their competitor's platform. IBM has a very nice Solaris to Linux porting guide. Sun has extensive information about porting from HP/UX to Solaris. And, as expected, HP has a number of guides explaining how to port from Solaris to HP/UX and from HP/UX to Linux. Any or all of those sources can yield specific details which can help you out. Some of them offer migration tools which convert shell scripts and identify trouble spots.

We hope that you found this information useful and will think of Integrated Computer Solutions as your partner in Unix and Linux software development.

About ICS

Driven by the belief that the success of any software application ultimately depends on the quality of the user interface, Integrated Computer Solutions, Inc., The User Interface Company™, of Cambridge, MA, is the world's leading provider of advanced user-interface development tools and services for professional software engineers in the aerospace, petrochemical, transportation, military, communications, entertainment, scientific, and financial industries. Long recognized as the platform of choice for visually developing mission-critical, high-performance Motif applications, ICS' BX series of GUI builders has recently been expanded to provide a complete line of tools that accelerate development of Java™. ICS is the largest independent supplier of add-on products to the Qt multi-platform framework developed by Trolltech. Supporting the software development community since 1987, ICS also provides custom UI development services, custom UI component development, training, consulting, and project porting and implementation services. More information about ICS can be found at http://www.ics.com

Integrated Computer Solutions Incorporated

The User Interface Company™

201 Broadway
Cambridge, MA 02139
617.621.0060
info@ics.com

www.ics.com