Volume 6, Number 8 June 1994 $2.50

MUUG Lines

Manitoba UNIX® User Group

Newsletter of the Manitoba UNIX© User Group

Client-Server Conquers Enterprises

By John Chisholm

Client-server systems have been part of work-group and departmental computing since the late 1980s. Now, thanks to UNIX on large servers and advances in databases and network management, client-server systems are being elevated to the level of the enterprise. HP 9000s, IBM RS/6000s, and Pyramid MIServers running Oracle and Sybase are supporting mission-critical, client-server applications and networks that before were handled solely by mainframes.

Many business forces are driving the re-engineering of traditional information systems to client-server systems. Many companies specifically mention the following forces:

- Higher employee productivity and morale. Employees are empowered through direct access to information that lets them make more of their own decisions. The result is a heightened sense of self-worth, greater feeling of control over their jobs, and overall higher morale. Ultimately, employee efficiency and independence increase, and personnel turnover declines.
- More competitive response to market changes. As markets dictate new products and changes to existing products, information systems must be able to respond and adapt at the same pace. To avoid delaying product introductions, information systems must be easily developed or modified and rapidly deployed.
- Better customer service. When employees who deal directly with customers have the required customer information at their fingertips, the employees provide more timely and appropriate service. More satisfied customers mean repeat business and more referrals to new customers.
- Commitment to total quality. Total quality programs center on ensuring that employees can both get and provide the information necessary to make quality decisions.

Traditional information systems are changing to client-server in two ways: without, or with, an accompanying change in business processes. In the first case, called information systems re-engineering (ISR), the applications supporting existing business processes, such as inventory control or order processing, are enhanced with client-server techniques, but the processes themselves do not change. For example, PCs with graphical user interfaces (GUIs) may replace character terminals, and distributed database servers may replace minis or mainframes, but tasks performed, organizational charts, and head counts do not change significantly. This approach lets enterprises enjoy the benefits of client-server in a matter of calendar quarters, rather than years. A minimum of buy-in from senior management and departments throughout the enterprise is required.

John Chisholm is president of John Chisholm Co. (Menlo Park, CA), a consulting firm specializing in marketing, strategy, alliances, and distribution for computer and software firms. You can reach him at 570-0792@MCIMail.

This Month's Meeting

Meeting Location:

Our June meeting is scheduled for Tuesday, June 14, at 6:30 PM. This meeting will be the traditional MUUG June BBQ. This year, Roland Schneider is hosting it at his home in Selkirk. A map is included in this month's newsletter.

Meeting Agenda:

Eat, drink, and be merry – but no computer talk!

Inside This Issue

Printing Courtesy of Xerox Canada Ltd.

Andrew Trauzzi's Newsletter?

By Andrew Trauzzi

The other day, a couple of MUUG members at work were commenting on the MUUG newsletter. They said it funny how almost all the articles in the newsletter were written by, or submitted by me. Unfortunately, this is true. Although I enjoy writing and looking for interesting articles to include in the newsletter, I never expected to have this "coverage."

Original Intent

When I took over as MUUG newsletter editor, Gilbert warned me that finding people to write articles would be more difficult than performing surgery on myself. For the first few issues, I was becoming comfortable with the editing routine, and didn't miss extra articles — after all, people didn't know what to submit. In the new year, however, I found myself researching, writing, and/or typesetting almost every article. Needless to say, I tried to find volunteers but almost everyone didn't know what to write about, or didn't have the time to write an article.

J. Random Hacker?

I realize that the poor economy has taken a toll on all of us. We are all working more overtime, have less time for our families, and are under more stress then ever before. I am prone to all of the above, and barely have time to complete the newsletter. Just last month, I submitted my final copy to Rory for printing, He ran off 200 or so and brought them over for stuffing. Earlier on in the week, I didn't know the name of the speaker from Hewlett-Packard, so I inserted a dummy name that I always notice — J. Random Hacker. Well I almost always notice it. Unfortunately, my error caused Rory to reprint and stuff all 200 newsletters by hand. (Thanks Rory!)

The Point

I certainly don't want my comments interpreted as whining and blathering, because I really enjoy editing this newsletter. Instead, I would like to ask all of you to consider writing or submitting an article for the newsletter. The next issue comes out in September, so the deadline for article submission is August 21st. That gives all of you over 2 months to find something interesting to write about. Hopefully, next year's newsletter will reflect the interests of the group, and not just one individual. In any case, have a great summer, and I will see you in September.

The 1993-1994 Executive

Copyright Policy and Disclaimer

Advertising Rates

Quarter page	$50
Half page	$75
Full page	$100
Insert (1-4 pages)	$100

Above prices are per issue. The first ad is charged at the full price; each successive month is 1/2 price.

Ad copy must be submitted by the final copy deadline for an issue (usually 3 weeks prior to the monthly meeting) in a format acceptable to the editor. (Please make arrangements with editor beforehand.)

Internet E-mail: editor@muug.mb.ca

Group Information

The Manitoba UNIX User Group meets at 7:30 PM the second Tuesday of every month, except July and August. Meeting locations vary. The newsletter is mailed to all paid-up members one week prior to the meeting. Membership dues are $25 annually and are due as indicated by the renewal date on your newsletter's mailing label. Membership dues are accepted at any meeting, or by mail.

Manitoba UNIX User Group
P.O. Box 130, Saint-Boniface
Winnipeg, Manitoba R2H 3B4

Internet E-mail: membership@muug.mb.ca

Reflections ...

By Bary Finch

... or is that "refractions"? Oh well. I'm just looking back on my year as MUUG President. I guess it would be refractions if I left a lot of myself behind. I seem to still be complete, so reflections may be the better term.

I have most certainly enjoyed this last year as MUUG President. I have gotten to know so many more of you before and after meetings, and through the questions that you forward to me via phone or email. Of course I can't really answer anything (it's tough being a figurehead) and I pass it on to the people that REALLY know what's going on.

Just kidding. Although I have gotten to know much more about MUUG over the last year. All the people in the executive from previous years make it easy to understand how MUUG runs, and what can be done to improve it. And all the new people bring in fresh ideas that keep our group following the leading trends in the industry.

I feel we have managed to make significant change over the last year. We now have a formal schedule of the year's presentations prepared early, and make sure we arrange for the best possible speakers from whatever the best source is. This source is now often one of the many vendors, both in town and out of town.

Our relationships with the vendors, and with the overall Winnipeg business community, has developed extensively with our Corporate Sponsorship program. This has established much more prominence for MUUG with local businesses. And they have shown their belief in us with their much appreciated support.

One reflection (there's that word again) of our effort to use the support of the sponsors, and return it to the local community, is in the kind of presentations that we are providing. Our topics are of interest for many of the local data processing professionals, judging by the range of people that attend our meetings.

We also are still providing the information that the casual user finds interesting. With the strong representation in our membership that the hobbyist UNIX user represents, MUUG continues to focus on what kinds of topics are appropriate for all UNIX users.

I would like to extend my thanks for everyone who attended the last meeting, and took the time to fill out our questionnaire on what topics you find interesting for the coming year. Without your input the MUUG executive has to take its best guess as to what you want to see presented at the meetings. Thanks again for helping us provide what you want. Another great development has been the Special Interest Group (SIG) for Linux and System Administration. This has continued strongly over the last year, with a number of technical presentations being provided. We will work to deliver more presentations on System Administration topics, with a focus on showing how it can be implemented in Linux.

If you've looked outside lately, you'll notice it's summer. Although it was 2 degrees the morning I wrote this. But we are officially into doing summer activities now, so it must be time for the annual MUUG barbecue. Elsewhere in this issue are the details of how to get to the barbecue. Please remember it starts at 6:30, not 7:30, so no need to stay home starving and waiting to eat!

With the barbecue, MUUG again ends its agenda for the summer, and we take July and August off. Don't fear, we'll be back for the September meeting, once again at the St.Boniface Research Center. In the mean time, have a great summer!

CORPORATE SPONSORS

The Manitoba UNIX User Group

gratefully acknowledges the generous support of the following

Corporate Sponsors

IBM

digital™

HEWLETT PACKARD

Great-West Life Assurance Company

XEROX *The Document Company*

TANDEM COMPUTERS

ONLINE BUSINESS SYSTEMS

C++ Q&A

By Marshall P. Cline

Question 34: What if I forget the '`[]`' when 'delete'ing arrays allocated via '`new X[n]`'?

Life as we know it suddenly comes to a catastrophic end.

It is the programmer's —not the compiler's— responsibility to get the connection between `new[]` and `delete[]` correct. If you get it wrong, neither a compile-time nor a run-time error message will be generated by the compiler.

Heap corruption is a likely result.

SECTION 9: Debugging and error handling

Question 35: How can I handle a constructor that fails?

Constructors (ctors) do not return any values, so no returned error code is possible. The best way to handle failure is therefore to 'throw' an exception.

If your compiler doesn't yet support exceptions, several possibilities remain. The simplest is to put the object itself into a 'half baked' state by setting an internal status bit. Naturally there should be a query ('inspector') method to check this bit, allowing clients to discover whether they have a live object. Other member functions should check this bit, and either do a no-op (or perhaps something more obnoxious such as '`abort()`') if the object isn't really alive. Check out how the iostreams package handles attempts to open nonexistent/illegal files for an example of prior art.

Question 36: How can I compile-out my debugging print statements?

This will NOT work, since comments are parsed before the macro is expanded:

```
#ifdef DEBUG_ON
   #define  DBG
#else
   #define  DBG //
#endif DBG cout << foo;
```

This is the simplest technique:

```
#ifdef DEBUG_ON
   #define  DBG(anything)  anything
#else
   #define  DBG(anything)  /*nothing*/
#endif
```

Then you can say:

```
//...
DBG(cout << "the value of foo is " << foo << '\n');
```

Any commas in your '`DBG()`' statement must be enclosed in a '`()`':

```
DBG(i=3, j=4);//<-- C-preprocessor error message generated
DBG(i=3; j=4);//<-- ok
```

There are also more complicated techniques that use variable argument lists, but these are primarily useful for '`printf()`' style (see question on the pros and cons of `<iostream.h>` as opposed to `<stdio.h>` for more).

SECTION 10: Const correctness

Question 37: What is 'const correctness'?

A program is 'const correct' if it never mutates a constant object. This is achieved by using the keyword '`const`'. Ex: if you pass a String to a function '`f()`', and you wish to prohibit '`f()`' from modifying the original String, you can either pass by value:

```
void  f(String  s)  { /*...*/ }
```

or by constant reference:

```
void  f(const String& s)  { /*...*/ }
```

or by constant pointer:

```
void  f(const String* sptr)  { /*...*/ }
```

but *not* by non-const ref:

```
void  f(String& s)  { /*...*/ }
```

nor by non-const pointer:

```
void  f(String* sptr)  { /*...*/ }
```

Attempted changes to 's' within a fn that takes a '`const String&`' are flagged as compile-time errors; neither run-time space nor speed is degraded.

Question 38: Is 'const correctness' a good goal?

Declaring the 'constness' of a parameter is just another form of type safety. It is almost as if a constant String, for example, 'lost' its various mutative operations. If you find type safety helps you get systems correct (especially large systems), you'll find const correctness helps also.

Short answer: yes, const correctness is a good goal.

Question 39: Is 'const correctness' tedious?

Type safety requires you to annotate your code with type information. In theory, expressing this type information isn't necessary — witness untyped languages as an example of this. However in practice, programmers often know in their heads a lot of interesting information about their code, so type safety (and, by extension, const correctness) merely provide structured ways to get this information into their keyboards.

Short answer: yes, const correctness is tedious.

Question 40: Should I try to get things const correct 'sooner' or 'later'?

Back-patching const correctness is *very* expensive. Every '`const`' you add 'over here' requires you to add four more 'over there'. The snowball effect is magnificent — unless you have to pay for it. Long about the middle of the process, someone stumbles on a function that needs to be const but can't be const, and then they know why their system wasn't functioning correctly all along. This is the benefit of const correctness, but it should be installed from the beginning.

Short answer: **Const correctness should not be done retroactively!!**

Question 41: What is a 'const member function'?

A const member function is a promise to the caller not to change the object. Put the word '`const`' after the member function's signature; ex: `class X { //... void f() const; };`

Some programmers feel this should be a signal to the compiler that the raw bits of the object's '`struct`' aren't going to change, others feel it means the *abstract* (client-visible) state of the object isn't going to change. C++ compilers aren't allowed to assume the bitwise const, since a non-const alias could exist which could modify the state of the object (gluing a '`const`' ptr to an object doesn't promise the object won't change; it only promises that the object won't change ***via that pointer***).

I talked to Jonathan Shopiro at the *C++AtWork* conference, and he confirmed that the above view has been ratified by the ANSI-C++ standards board. This doesn't make it a 'perfect' view, but it will make it 'the standard' view.

Dr. Marshall P. Cline is the founder and President of Paradigm Shift, Inc., a firm that specializes in on-site training for C++, OOD, OOA, consulting, and reusable/extensible C++ class libraries. For more information, send e-mail to "info@parashift.com".

UNIX Q&A

Originally Compiled by Ted Timar

Submitted by Andrew Trauzzi

Question 1: How do I find the creation time of a file?
You can't — it isn't stored anywhere. Files have a last-modified time (shown by "`ls -l`"), a last-accessed time (shown by "`ls -lu`") and an inode change time (shown by "`ls -lc`"). The latter is often referred to as the "creation time" — even in some man pages — but that's wrong; it's also set by such operations as `mv`, `ln`, `chmod`, `chown` and `chgrp`.
The man page for "`stat(2)`" discusses this.

Question 2: How do I use "rsh" without having the rsh hang around until the remote command has completed?
The obvious answers fail:

```
rsh machine command &
        or
rsh machine 'command &'
```

For instance, try doing

```
rsh machine 'sleep 60 &'
```

and you'll see that the '`rsh`' won't exit right away. It will wait 60 seconds until the remote '`sleep`' command finishes, even though that command was started in the background on the remote machine. So how do you get the '`rsh`' to exit immediately after the '`sleep`' is started?
The solution — if you use csh on the remote machine:

```
rsh machine -n 'command >&/dev/null </dev/null &'
```

If you use sh on the remote machine:

```
rsh machine -n 'command >/dev/null 2>&1 </dev/null &'
```

Why? "`-n`" attaches rsh's stdin to `/dev/null` so you could run the complete rsh command in the background on the **local** machine. Thus "`-n`" is equivalent to another specific "`< /dev/null`".

Furthermore, the input/output redirections on the **remote** machine (inside the single quotes) ensure that rsh thinks the session can be terminated (there's no data flow any more.)
Note: The file that you redirect to/from on the remote machine doesn't have to be /dev/null; any ordinary file will do.

In many cases, various parts of these complicated commands aren't necessary.

Question 3: How do I truncate a file?
The BSD function `ftruncate()` sets the length of a file. Xenix — and therefore `SysV r3.2` and later — has the `chsize()` system call. For other systems, the only kind of truncation you can do is truncation to length zero with `creat()` or `open(..., O_TRUNC)`.

Question 4: How do I set the permissions on a symbolic link?
Permissions on a symbolic link don't really mean anything. The only permissions that count are the permissions on the file that the link points to.

Question 5: How do I "undelete" a file?
Someday, you are going to accidentally type something like "`rm * .foo`", and find you just deleted "`*`" instead of "`*.foo`". Consider it a rite of passage.

Of course, any decent systems administrator should be doing regular backups. Check with your sysadmin to see if a recent backup copy of your file is available. But if it isn't, read on.

For all intents and purposes, when you delete a file with "`rm`" it is gone. Once you "`rm`" a file, the system totally forgets which blocks scattered around the disk comprised your file. Even worse, the blocks from the file you just deleted are going to be the first ones taken and scribbled upon when the system needs more disk space. However, never say never. It is theoretically possible *if* you shut down the system immediately after the "`rm`" to recover portions of the data. However, you had better have a very wizardly type person at hand with hours or days to spare to get it all back.

Your first reaction when you "`rm`" a file by mistake is why not make a shell alias or procedure which changes "`rm`" to move files into a trash bin rather than delete them? That way you can recover them if you make a mistake, and periodically clean out your trash bin. Two points: first, this is generally accepted as a *bad* idea. You will become dependent upon this behaviour of "`rm`", and you will find yourself someday on a normal system where "`rm`" is really "`rm`", and you will get yourself in trouble. Second, you will eventually find that the hassle of dealing with the disk space and time involved in maintaining the trash bin, it might be easier just to be a bit more careful with "`rm`". For starters, you should look up the "`-i`" option to "`rm`" in your manual.

If you are still undaunted, then here is a possible simple answer. You can create yourself a "can" command which moves files into a trashcan directory. In `csh(1)` you can place the following commands in the "`.login`" file in your home directory:

```
alias can 'mv \!* ~/.trashcan' # junk file(s) to trashcan
alias mtcan 'rm -f ~/.trashcan/*' # irretrievably empty trash
if ( ! -d ~/.trashcan ) mkdir ~/.trashcan
# ensure trashcan exists
```

You might also want to put a:

```
rm -f ~/.trashcan/*
```

in the "`.logout`" file in your home directory to automatically empty the trash when you log out. (sh and ksh versions are left as an exercise for the reader.)

MIT's Project Athena has produced a comprehensive delete/undelete/expunge/purge package, which can serve as a complete replacement for rm which allows file recovery. This package was posted to comp.sources.misc (volume 17, issue 023-026)

Question 6: How can a process detect if it's running in the background?
First of all: do you want to know if you're running in the background, or if you're running interactively? If you're deciding whether or not you should print prompts and the like, that's probably a better criterion. Check if standard input is a terminal:

```
sh: if [ -t 0 ]; then ... fi C: if(isatty(0)) { ... }
```

In general, you can't tell if you're running in the background.

The fundamental problem is that different shells and different versions of UNIX have different notions of what "foreground" and "background" mean — and on the most common type of system with a better-defined notion of what they mean, programs can be moved arbitrarily between foreground and background!

UNIX systems without job control typically put a process into the background by ignoring SIGINT and SIGQUIT and redirecting the standard input to "`/dev/null`"; this is done by the shell.

Shells that support job control, on UNIX systems that support job control, put a process into the background by giving it a process group ID different from the process group to which the terminal belongs. They move it back into the foreground by setting the terminal's process group ID to that of the process. Shells that do *not* support job control, on UNIX systems that support job control, typically do what shells do on systems that don't support job control.

GNU Review

By Peter Graham

Still improving — I'll beat the procrastination thing yet. I'm writing this article one whole week before the deadline. Yes, I am looking for a pat on the back. *(ok Peter — pat, pat, pat — ed.)* Alternatively, someone can "buy" me a beer at the BBQ. Don't get too generous though because I'm only doing this because I will be out of town when the article is really due. ;->

Gnu make - A better make program

This month we'll talk about Gnu make. For convenience and to avoid confusion, we'll refer to it as 'gmake'. Gnu make is an enhanced version of the standard make utility which is based on the version of make which is distributed with the latest Berkeley release. It offers many features not available in older makes and incorporates the best of the features from many makes.

Conventional make and gmake perform the same basic function. Quoting from the online info that comes with gmake:

"The 'make' utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them. This manual describes GNU 'make', which was implemented by Richard Stallman and Roland McGrath. GNU 'make' conforms to section 6.2 of 'IEEE Standard 1003.2-1992' (POSIX.2).

You can use 'make' with any programming language whose compiler can be run with a shell command. Indeed, 'make' is not limited to programs. You can use it to describe any task where some files must be updated automatically from others whenever the others change."

The difference is gmake allows you to do more complicated (and useful) things easily.

gmake features

I will now describe what makes gmake better than your average make program.

- **VPATH variable** — An undocumented features from SysV make. Allows large software systems to be housed in multiple directories easily without having to explicitly specify relative pathnames. VPATH says where to search for files involved in rules.
- **included makefiles** — Makefiles may include other makefiles. This makes it easy to write "libraries" of makefiles containing actions for specific purposes.
- **integrated environment variables** — Make variables may be read from the environment.
- **passed options** — Options to make are automatically passed through the variable MAKEFLAGS to recursive invocations of make.
- **numerous builtin variables** — Builtin variables exist for many useful things.
- **pattern rules** — Rules which do pattern matching using '%'.
- **parallelism** — To enhance the speed of large rebuilds and much, much, more!

gmake requirements

Gnu make runs on a wide variety of platforms. It likes to be compiled with gcc but this is not a hard and fast requirement.

gmake installation

Guess what? Its a Gnu install so... "`./configure; make; make install`" and you're done. Its as easy as that! Some people may want to keep their old make around (perhaps for sentimental reasons ;->). If so, it's easy to install Gnu make as gmake either manually after the fact or by editing the Makefile. Note that if you do this, you will probably also want to fix up the placement of the man page and the info files.

gmake Usage

Using gmake can be as easy as using make. Thus, if you simply have an existing make file, you say "`make`". On the other hand, if you want to create your own makefiles you can use all of gmake's features. The space limitations (not to mention the time limitations) preclude my discussing the use of the many features of gmake. A hint of what can be done is given in the section on gmake features. See the info files (via the info program which is also available on `prep.ai.mit.edu`) for detailed information.

gmake Summary

Name	make
Description	Replacement for standard Unix make (derived from the new Berkeley make). Tool to help automate the compilation of large software systems (among other things).
Archive Loc'n	prep.ai.mit.edu:`/pub/gnu/make-3.70.tar.gz /pub/gnu/make-doc-3.70.tar.gz`
Archive size	427221 bytes (code) 253635 bytes (doc'n)
Approx Space to Install	3.5MB
Time to Install (Sparc-1)	7.5 minutes
Pros	• Free, small, and easy to install. (as always) • Significant extensions to existing makes. • Backwards compatible with older makes (mostly).
Cons	• Relatively significant re-learning curve for people used to old makes...but • extensive "info" files come with it (I know, I know, "What's a PRO doing in with the CONs???")

See you all at the BBQ! That's it for this year. E-mail to pgraham@cs.umanitoba.ca.

The PowerOpen White Paper

Part 2 — The PowerOpen Association

Submitted by Keri Gustafson through Bary Finch

Last month's article focused on the hardware behind the PowerPC movement, the PowerPC RISC chip. This month's article will examine the structure that has been built around the PowerPC chip — the PowerOpen association.

Introduction

In 1991, the Apple, IBM and Motorola alliance announced the PowerOpen™ Environment initiative — the computing solution for the '90s and beyond. Since that time, those companies — along with Bull, Harris Corporation™, Tadpole Technologies™ and THOMSON-CSF™ — have been working to realize that initiative by defining an environment responsive to both customer and industry demands. Building on a foundation of open systems conformance, the environment is expected to give customers access to a large base of applications.

The PowerOpen Environment can be described as the combination of any PowerPC Architecture compliant processor together with its system software that is designed to help enable application binary compatibility across different vendors' PowerPC platforms. In addition, the PowerOpen Environment incorporates the hardware, system software, and application programming interfaces required to advance the goal of running both UNIX™ and Macintosh™ applications presented with the "look and feel" that users want.

Users from either the UNIX or Macintosh worlds would find their applications working in familiar ways. Whatever the interface, users would see and manipulate a common file structure. Data from either type of application could be cut and pasted to the other. The dream of running UNIX and native Macintosh applications on the same platform has moved closer to reality with the introduction of the PowerOpen Environment.

For UNIX users, the PowerOpen ABI will provide platform binary compatibility based upon leading edge hardware technology and software specifications. The goal of the PowerOpen specification is to enable binary applications to run across PowerOpen compliant systems from many vendors.

The PowerOpen Operating Environment is thus the next evolutionary step for both UNIX and Macintosh computing. To sum it up, the environment plans to provide a standards-driven, open environment for tomorrow's users today.

Endorsements from Vendors

Numerous vendors have already announced support for PowerOpen. System suppliers like Apple, Bull, Harris, IBM, Tadpole Technology and THOMSON-CSF are already committed to the PowerOpen Environment. As recognition of the PowerOpen grows, other suppliers are also expected to announce their support. System vendors, however, aren't the only ones voicing support for the PowerOpen Environment - software developers are also joining the ranks.

Several companies plan for their operating systems to comply with the Application Binary Interface(ABI) for the PowerOpen Environment. IBM and Bull have announced plans to market their compliant operating system as AIX/6000T, and THOMSON-CSF as UNI/XT(TM). In addition, Apple expects to market a compliant operating system. Each company will add its own extensions to their PowerOpen compliant implementation.

PowerOpen Association

The PowerOpen Association, a group organized and chartered to the advancement of the PowerOpen environment, supports the PowerOpen specification which includes an application binary interface, API (application programming interface; XPG4, XTI, XNFS and the PowerPC RISC microprocessor). The PowerOpen Environment is designed to help enable software developers to produce "shrink-wrapped" applications that will run on multiple platforms. The ABI documentation will cover the PowerOpen Environment basics, porting and migration, Macintosh Application Services, PowerPC Architecture, and Power PC 601 User's manual.

Foundations of the PowerOpen Environment

Applications expected for the PowerOpen environment are comprised of many existing applications as well as new applications that will likely be developed to fully exploit the PowerOpen specification's potential.

The PowerOpen Environment will allow users to work simultaneously with graphical applications based on a Macintosh or OSF/Motif-based interface, and with character-based applications. No matter what the user interface, each application will ride on the PowerOpen Environment which is platform and I/O independent. The goal of the PowerOpen Environment specification, is to enable software developers not to have to take platform-specific functions and I/O bus dependencies into consideration. Thus, software developers could concentrate their efforts on the functionality of end-user applications, without the trouble of dealing with hardware-specific issues.

The PowerOpen Environment is designed to help enable software vendors to produce "shrink-wrapped software" and powerful server systems. An important goal of the PowerOpen Environment is to offer users the same convenient access to software that Macintosh and PC-compatible users have enjoyed for years.

The Hardware: The PowerPC Behind the PowerOpen Environment

The PowerOpen Environment runs on the evolutionary, high-performance RISC microprocessor technology: PowerPC. The PowerPC Architecture, developed by IBM and Motorola with input from Apple, derives from IBM's successful Performance Optimized With Enhanced RISC (POWER™) architecture. ☛

The PowerPC chip makes use of both IBM and Motorola's world-class chip design and fabrication techniques and facilitates a truly scalable processor family. Consequently, the PowerPC microprocessor is designed to support computers ranging from pen-based systems to desktop PCs to multiprocessing servers to multiprocessor super computers, including real-time systems and server systems. The 601 and 604 microprocessors will give desktop designers a chip for office computing and will have extensive support for multiprocessing. At the same time, the 603 microprocessor will give system suppliers the chip needed for low-end desktop computers and laptops. The 620 microprocessor will supply the muscle needed for high-end workstations, servers, and multiprocessor systems.

Application Binary Interface (ABI)

The Application Binary Interface (ABI) defines the structure of the application in the PowerOpen Environment. This includes such key definitions as loading and linking, conventions, object formats, the execution environment, networking infrastructure, and installation and packaging information.

Application Programming Interface (API)

The Application Programming Interface defines the set of system calls, library functions, header files, commands and utilities that an application developer is allowed to use to develop a compliant application. ISVs are mainly concerned with programming to the API. Underlying the PowerOpen API are key industry standards such as:

- XPG4
- XNFS
- XTI
- X11R5
- AES
- POSIX

The Environment definition will define all the elements that developers need to create applications that will install, execute, and run unmodified on multiple platforms conforming to the PowerOpen Environment.

The networking API provides the commands and parameter-passing definitions for inter-system operations. Both streams and sockets are used for networking in the PowerOpen Environment. TCP/IP is one of the underlying protocols used for networking, but any streams-and-sockets-compatible networking protocols could be used. We anticipate that our members will submit candidates for future inclusion. System management in the PowerOpen Environment will initially consist of install and update functions for applications.

The windowing extensions for the PowerOpen Environment consists of a window manager, application commands and parameters, and the communication protocol. The PowerOpen windowing system is derived from the X Window System Release 11 Version 5 (X11R5), which provides a client/server based graphical windowing system.

Macintosh Application Services Apple Macintosh applications run through the Macintosh application-services extension. Macintosh Finder, the Macintosh desktop creator and manager, runs within a single X Window. Both PowerPC and 680x0 Macintosh applications are expected to run simultaneously from the same system.

The Macintosh Finder plans to provide the familiar Macintosh Desktop Graphical User Interface within an X Window on the PowerOpen system. All files (including non-Macintosh documents and applications) would appear as icons; users would simply double-click on a Macintosh or UNIX icon to open a file or launch an application. This brings Macintosh point-and-click simplicity to UNIX application users. The Finder will be based on Macintosh System 7 system software.

The Macintosh Application Engine is planned as the heart of the Macintosh Applications Services. The Macintosh Application Engine plans to maximize the speed of running Macintosh 680x0 applications on a PowerOpen system. All Macintosh 680x0 applications are intended to be supported by an emulator, which interprets the 680x0 code to instructions usable by the PowerOpen platform. The system is designed to minimize the time spent in the emulator, and maximize the time spent executing application commands in native PowerOpen code. The goal is for Macintosh application performance to be much faster.

"The PowerOpen Environment incorporates a rich set of open computer standards"

Macintosh applications — both Macintosh 680x0 applications, and Macintosh PowerPC applications - plan to interact with the PowerOpen system layer through the Macintosh Toolbox, an interface written in native PowerOpen code. Since most Macintosh 680x0 applications spend up to 90% of their processing time in the Macintosh Toolbox, this should greatly improve the speed of running Macintosh applications on the PowerOpen platform.

International Language Support

The PowerOpen Environment incorporates a rich set of open computer standards including international language support. Character representation is handled not only by UNIX's standard 7-bit ASCII, but also by the ISO 8859 family of 8-bit extended ASCII code sets, as well as the de facto standard PC code set (IBM-850). For Asian languages, character encodings are supported by the Extended UNIX Code set. Extended UNIX Code support is also provided for Chinese, Japanese, and Korean characters.

Besides this extensive character support, language-customs and conventions support is provided for the world's most widely used languages, including Chinese, English, French, German, Japanese, and Spanish. Additionally, the same tools that are used to develop the language support are provided as a part of the environment. This feature allows further customization of customs and conventions. Optionally, software developers may also choose to develop additional language support. What this all means is that the PowerOpen Environment, from the very beginning, is ☛

designed for internationalization and not limited to one language.

XCOFF In addition to application development programming considerations, binary compatibility requires definition of a consistent output format. The PowerOpen Environment defines this to be the extended common object file format (XCOFF). XCOFF provides the object file definition for applications. Moreover, XCOFF encompasses shared libraries and dynamic linking. This means that software developers can deliver compact, memory-efficient programs.

Benefits for Software Developers

The PowerOpen ABI specification gives the environment several advantages over other operating systems and environments. First, because of its open programming interface, software developers will find the PowerOpen Environment both attractive and easy to write to. A related factor in PowerOpen's favor is that, unlike all other mainstream operating environments, the PowerOpen Environment is extendible through an open process. Software developers will not be caught flat by unannounced changes.

The PowerOpen Environment goal of cushioning software developers from having to deal with hardware specifications also is another reason it is easy for them to write to the PowerOpen specification. Its scalability and its reliance on industry standards are factors that weigh heavily in PowerOpen favor. The PowerOpen operating environment has obvious advantages for software developers will help to encourage its quick adoption as the environment of choice for software developers, large and small.

"The PowerOpen Environment is extendible through an open process"

The System Verification Test Suite (SVTS) will verify that the platform is compliant with the PowerOpen specification. This will ensure that ISVs who develop to the PowerOpen Environment definition need only develop a single port to have their application run on multiple vendor's PowerOpen compliant systems. This will result in significant savings of reduced porting and maintenance costs.

The Future

The PowerOpen operating environment offers many advantages that will help ensure its success. It is derived from proven technology and is supported by major suppliers. It expects to give customers access to a large base of applications. The PowerOpen specification is designed with the goal of supporting "shrink-wrapped" applications across compatible platforms, making it attractive to applications-software developers. It is built to be upwardly compatible and the scalable architecture will allow the PowerOpen environment to expand to meet computing demands well into the next century. The first PowerPC processor is available today and will be quickly followed by the rest of the single-chip processors that span the full range from 32-bit (energy efficient, cost effective) chips to 64-bit multi-processor enabled chips.

Emerging needs are met by having designers of technologies complying with the ABI committed to the continuous improvement of both its hardware and software elements. This continuing evolution will be accomplished by maintaining constant communications with all PowerOpen Environment adopters, system vendors, software developers, and customers, through their participation in the PowerOpen Association.

Promoting application availability on the PowerOpen Environment is another goal of the PowerOpen Association. The PowerOpen Environment is designed to be able to run DOS, Microsoft Windows, OS/2 and other applications through third party software vendor's simulators. As technologies such as binary-to-binary converters mature, however, these applications may be ported to run in native mode on the PowerPC platforms.

The PowerOpen Environment will not evolve merely in a reactive fashion. The PowerOpen Association's Technical Working Groups will be constantly planning the PowerOpen specification's evolution two versions up the road. Enhancements to the PowerOpen Environment will be recommended through the PowerOpen Association.

As the PowerOpen operating environment evolves, the maintenance of binary compatibility with existing hardware and software will be an important consideration and a primary goal of the Association. The Association will also work with standards bodies like X/Open as appropriate. The PowerOpen Association will review new technologies for inclusion in the PowerOpen Environment such as object-oriented frameworks and multimedia API's. PowerOpen is truly a system for today and tomorrow.

The PowerOpen Association
25 Burlington Mall Road
Burlington, MA 01803
Phone: (800) 457-0463 U.S & Canada
(617) 273-1550 International

Portions of this document were provided by International Business Machines.

Annual MUUG Barbecue

June 14, 6:30 pm

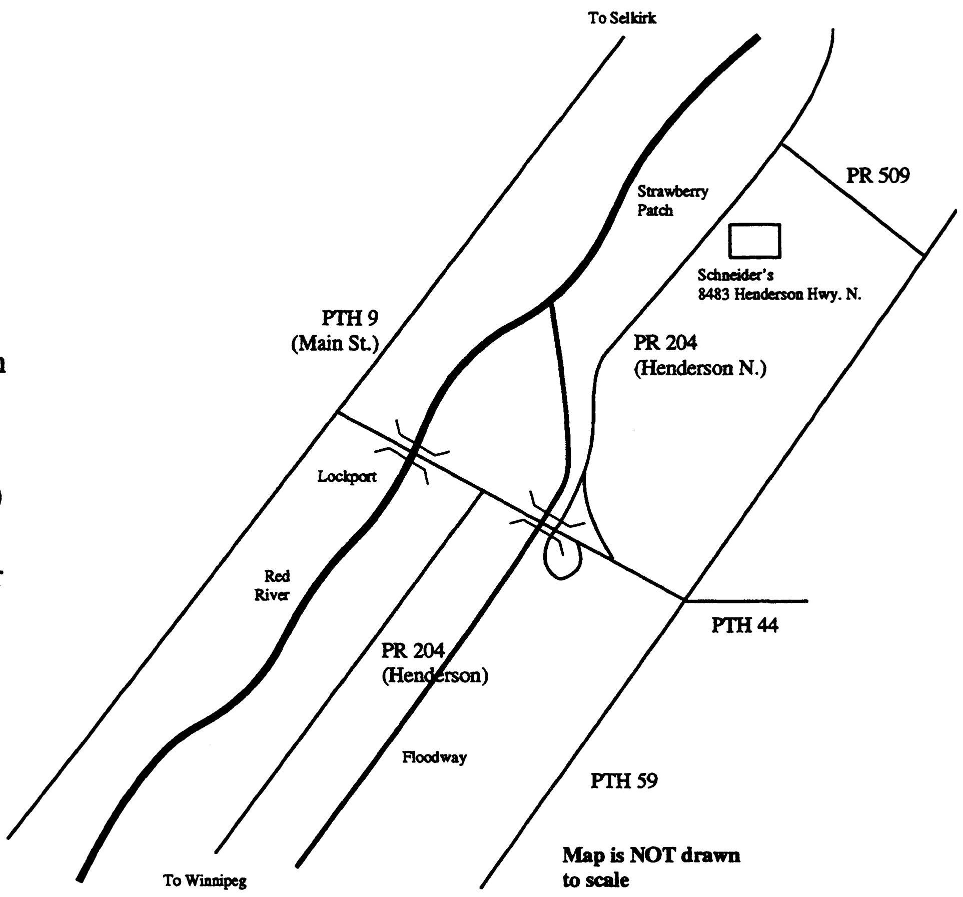
Host: Roland Schneider
phone: 1-482-5173

Where: The Schneiders'
8483 Henderson Hwy
N. (PR 204)
Across from *The Strawberry Patch*
About 20 minutes from the Perimeter
(See map)

When: Tuesday, June 14, 6:30 pm
(1 hour earlier than our normal meetings)

Bring: Meat to cook
Beer
Lawn chairs

RSVP: Roland Schneider
1-482-5173 (days and evenings)
e-mail: <rsch@muug.mb.ca>
or
Andrew Trauzzi
986-3898 (days)



We will supply chips and other nibble food, soft drinks, salads, and a cake. (and insect repellent if necessary). Spouses or significant others, and children, are also welcome. Bring a swimsuit if you want to take a dip in our pond.

Sig Sideline

By Brad West, Sig Co-ordinator

For our last SIG meeting of the season (May 17th), we had a good turnout. The evening started with a surprise presentation by Michael Doob. Michael demonstrated his working Linux system installed on a 386sx 16 MHz laptop with 4 MB of ram and a 20 MB hard drive. He did a run- through of the installation process, and the system worked amazingly well. Linux was installed to a bare minium and took up only 8 MB of the hard disk and had a 4 MB swap disk. This left the system with 40% of the hard drive for applications (8 MB). The system was demonstrated working with TeX., and did a good job (considering the hardware limitations). Michael demonstrated the installation process he used for this system by using the bare boot disk, file system disk, and the a1 — a3 disks. The Linux distribution used on this system was Slackware 1.2.0. The rest of the evening continued with the round table format.

That's it for this year, I hope it was a enjoyable and learning experience for all. I would like to give a special thanks to ISM and Wolfgang von Thuelen for the use of ISM premises for our meetings this year. Also, thanks to all who presented topics this year. We plan to have more presentation setup for next year's meeting, so if anyone is interested in being a guest speaker at a SIG meeting, or you have a specific topic of interest, let me know. I can be reached by email <bwest@muug.mb.ca> or my work phone is 983-0336. The presentation for next year's meeting is TBA. The next meeting is tentatively scheduled for Tuesday, September 13, at 7:30 PM. This meeting will again be held at ISM, 400 Ellice Avenue, behind Portage Place. Our host is Wolfgang von Thuelen. He will be waiting in the lobby as of 7:15 PM to let everyone in. Wish everyone a great summer and hope to see you next year.

Coming Up

Meeting:
September's meeting is scheduled for Tuesday, September 13, at 7:30 PM. Meeting location will be the St-Boniface Research Centre, as usual. Stay tuned for details, and have a great summer!