Volume 6, Number 7 May 1994 $2.50

MUUG Lines

Manitoba UNIX® User Group

Newsletter of the Manitoba UNIX© User Group

Is UNIX Secure Enough?

By Andrew Trauzzi

A few months ago, a story appeared in the news about an Automated Teller Machine scam. The thieves had constructed a fake ATM and placed in a local shopping mall. When someone placed their card in the machine and attempted to withdraw some money, the machine displayed a message that the network was down and the transaction cancelled. What the machine actually did was send the electronic information and the user's password to the thieves who were waiting with a magnetic encoder. They encoded the information on a card, inserted it into a real ATM, entered the password, and withdrew the maximum amount of money they could from the account.

When I read that, it reminded me about the fake login screen scam — users would type their login name and password into a program that emulated the real login screen. The user information was recorded, and the program proceeded to actually log the user in. So how can we protect ourselves from fake ATMs and login screens? Well, if the criminals are smart and thorough enough, it's very difficult. New high-tech crimes are one of the spin-offs for living in a "globally-connected" world. UNIX systems play a large role in this global-connection, and as such, are subject to incredible scrutinization and attempted break-ins.

Is UNIX really that insecure? Probably not — according to the experts. UNIX is just well-understood by many more people than MVS or other mainframe operating systems. Campuses all over the world are training grounds for hackers and other students anxious to see how far they can take their new-found knowledge. Should we stop training computer students in UNIX? I don't think so, unless we want to produce professionals that lack the necessary skills to function in today's workplace. The answer seems to currently lie in proactive and reactive security measures.

UNIX machines are extremely well-connected. This connectivity, along with inexperienced system administration, allows UNIX machines to be constantly under 'hack attack'. But connectivity also gives UNIX system administrators the ability to quickly share information that allows security 'holes' to be reported and filled very quickly. In fact, there is an organization called the Computer Emergency Response Team (CERT) set up by DARPA, whose sole function is to collect and distribute computer security breach reports, and security tools. CERT can be reached at (412) 268-7090 or `<cert@cert.sei.cmu.edu>`.

If your company is thinking of purchasing UNIX machines, or connecting to the Internet, then there are a number of precautions you can take to maintain a high level of security. All center around proper system administrator and user education, and keeping up to date with security news. There are many books you can purchase on UNIX security as well, but they are obviously ineffective if the knowledge is not put to practical use.

This month's speaker is J. Random hacker of hacking industries. He will be discussing some common (and not so common) security techniques used by large companies. Hope to see you there!

This Month's Meeting

Meeting Location:
Our next meeting is scheduled for Tuesday, May 10, at 7:30 PM. Once again, the meeting will be held in the auditorium of the St-Boniface Hospital Research Centre, just south of the hospital itself, at 351 Taché. You don't have to sign in at the security desk — just say you're attending the meeting of the Manitoba UNIX User Group. The auditorium is on the main floor, and is easily found from the entrance.

Meeting Agenda: See inside for details.

Inside This Issue

Printing Courtesy of Xerox Canada Ltd.

RAMBLINGS

Is There A Software Crisis?

By Andrew Trauzzi

I am one of a number of lucky Canadians to be chosen to take part in a Federal Government job survey. :¬(Last month, the woman responsible for collecting job information began entering her data into a computer program — unfortunately written to lay off unnecessary clerks. The Government gave them all portable computers with modems so they could easily transmit their information from home to Stats Canada. Sounds great, right? Well, in the ever-amazing PC world, great ideas can sometimes end up costing more then performing the same job by hand.

Now I won't go into the usual, "why should users have to know about config.sys, or system.ini?" Instead, this problem lies with the software developers themselves. First, the survey took ten minutes longer then it did before, because when the collector made a mistake, she had to press Alt-F10 to clear ALL the information entered on the screen! I asked her to read out the instructions that came with the program, and sure enough, if you make a mistake and leave the field, you have to start all over again! That was one of a large number of annoyances that this program gave her, and by the time she was finished asking questions, both her and I were ready to throw her PC out the window! Should people be forced to use (and hate) computers this way?

With the advent and proliferation of Windows for the PC, a whole slew of incredibly bad programs have appeared for the PC. I would much rather have a command-line interface then be forced to jump through gooey hoops in order to perform some simple operation. Don't get me wrong, I have used and programmed Macs and Windows PCs for over eight years now, and I love EFFECTIVE GUIs. I just think that some programmers should be forced to use their own 'creations'.

Another case in point. A large company decided to convert a large application from MVS to Windows. In order to 'reduce the amount of training and stress on employees', this company has decided to forgo all the advantages of a GUI environment and make all their windows exactly like their CICS screens — FKeys and all! Yikes! Someone should introduce this company to the '80s.

So, if there are any programmers or designers reading this (and I think there are), please put some thought into your GUI design, and help stop people from hating computers. ➻

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

So Many Choices

By Bary Finch

With our current "season" drawing quickly to a close, we are now faced with the fun of choosing appropriate topics for our 1994 / 1995 meeting schedule. There seems to be an almost infinite level of choice, even when we continue to focus on UNIX based technology, as we of course will.

The executive has already had one meeting to try and get a preliminary list of topics going. We came up with a rather extensive choice of possibilities, however we will have to be realistic and get it down to the eight topics we really need. It was felt we should not really follow a "theme" for the year, as the "hot" topics are across too broad a range. It would be a disservice to decide on a theme that might exclude a very popular topic that we could easily get a speaker for. So we will proceed with trying to determine the best of the list we have, and start putting together the 1994 / 1995 program.

It's Your Choice

Here is where you, the members, can help us choose what you want. We are going to prepare a list of the topics we have come up with, and present this list to you. This will probably be at the next meeting on May 10. We would like you to let us know your priority on the possible topics by ranking which you would prefer to see. This is valuable and necessary information for your executive to make the right decisions in our meeting topics.

We have already received some suggestions for topics from members. Thank you to those that have contributed. For those of you who know specific topics you would like to see, but haven't let us know yet, please take the opportunity to add your suggestions to the list of topics that we want you to prioritize.

One dilemma that we are encountering is the depth of a general topic, and where to dive into it and get presenters. To explain that last cryptic sentence, let me use an example such as "Client / Server". Just exactly what does "Client / Server" mean? I know of dozens of definitions, and each must be qualified against your specific requirements. Unfortunately, in our case, our only requirement is to present useful information on the topic, so we can't determine an appropriate definition in this manner.

Elephant Lipstick?

So we must dive deeper into Client / Server and find specific aspects that are of interest to our members. It is so broad a range that it varies from nice GUIs on traditional mainframe applications (otherwise known as putting lipstick on the elephant) to distributed application systems that "seamlessly" interconnect with any other corporate resource necessary. Somewhere in that range is at least one good topic to present to MUUG!

Well, we will continue to work on this, and get our 1994 / 1995 program built so we include the best topics in the industry. We appreciate your input, so keep those cards and letters coming!

The Barbecue is Coming Up

As for this season, we still have our upcoming presentation on Security by Hewlett-Packard. And in June we will once again have our annual barbecue. Stay tuned for more details on the barbecue (you know, little details — like where it's being held).

See you on May 10!

CORPORATE SPONSORS

The Manitoba UNIX User Group

gratefully acknowledges the generous support of the following

Corporate Sponsors

IBM

digital™

HEWLETT PACKARD

UM

XEROX The Document Company

Great-West Life Assurance Company

ONLINE BUSINESS SYSTEMS

TANDEM COMPUTERS

C++ Q&A

By Marshall P. Cline

This month's column examines C++ stream operators, and introduces some freestore management concepts.

SECTION 7: Input/output via <iostream.h> and <stdio.h>

Question 26: How can I provide printing for a 'class X'?

Provide a friend operator <<:

```
class X {
public:
   friend ostream& operator<< (ostream& o, const X& x)
      { return o << x.i; }
   //...
private:
   int i;    //just for illustration
};
```

We use a friend rather than a member since the 'X' parameter is 2nd, not 1st. Input is similar, but the signature is:

```
istream& operator >> (istream& i, X& x);
//not 'const X& x' !!
```

Question 27: Why should I use <iostream.h> instead of the traditional <stdio.h>?

See next question.

Question 28: Printf/scanf weren't broken; why 'fix' them with ugly shift operators?

The overloaded shift operator syntax is strange at first sight, but it quickly grows on you. However syntax is just syntax; the real issues are deeper. Printf is arguably not broken, and scanf is perhaps livable despite being error prone, however both are limited with respect to what C++ I/O can do. C++ I/O (left/right shift) is, relative to C (printf/scanf):

- type safe — type of object being I/O'd is known statically by the compiler rather than via dynamically tested '%' fields.
- less error prone — redundant info has greater chance to get things wrong C++ I/O has no redundant '%' tokens to get right.
- faster — printf is basically an 'interpreter' of a tiny language whose constructs mainly include '%' fields. the proper low-level routine is chosen at runtime based on these fields. C++ I/O picks these routines statically based on actual types of the args.
- extensible — perhaps most important of all, the C++ I/O mechanism is extensible to new user-defined data types (imagine the chaos if everyone was simultaneously adding new incompatible '%' fields to printf and scanf?!). Remember: we want to make user-defined types (classes) look and act like 'built-in' types.
- subclassable — ostream and istream (the C++ replacements for FILE*) are real classes, and hence subclassable. This means you can have other user defined things that look and act like streams, yet that do whatever strange and wonderful things you want. You automatically get to use the zillions of lines of I/O code written by users you don't even know, and they don't need to know about your 'extended stream' class. Ex: you can have a 'stream' that writes to a memory area (incore formatting provided by the standard class 'strstream'), or you could have it use the stdio buffers, or [you name it...].

SECTION 8: Freestore management

Question 29: Does 'delete ptr' delete the ptr or the pointed-to-data?

The pointed-to-data.

When you read 'delete p', say to yourself 'delete the thing pointed to by p'. One could argue that the keyword is misleading, but the same abuse of English occurs when 'free'ing the memory pointed to by a ptr in C: free(ptr); /* why not 'free_the_stuff_pointed_to_by(p)' ?? */

Question 30: Can I free() ptrs alloc'd with 'new' or 'delete' ptrs alloc'd w/ malloc()?

No. You should not mix C and C++ heap management.

Question 31: Why should I use 'new' instead of trustworthy old malloc()?

malloc() doesn't call constructors, and free() doesn't call destructors. Besides, malloc() isn't type safe, since it returns a 'void*' rather than a ptr of the right type (ANSI-C punches a hole in its typing system to make it possible to use malloc() without pointer casting the return value, but C++ closes that hole). Besides, 'new' is an operator that can be overridden by a class, while 'malloc' is not overridable on a per-class basis (ie: even if the class doesn't have a constructor, allocating via malloc might do inappropriate things if the freestore operations have been overridden).

Question 32: Why doesn't C++ have a 'realloc()' along with 'new' and 'delete'?

Because realloc() does *bitwise* copies (when it has to copy), which will tear most C++ objects to shreds. C++ objects know how to copy themselves. They use their own copy constructor or assignment operator (depending on whether we're copying into a previously unused space [copy-ctor] or a previous object [assignment op]).

Moral: never use realloc() on objects of a class. Let the class copy its own objects.

Question 33: How do I allocate / unallocate an array of things?

Use `new[]` and `delete[]`:

```
      Thing* p = new Thing[100];
      //...
      delete [] p;
      //older compilers require you to use 'delete
      //[100] p'
```

Any time you allocate an array of things (ie: any time you use the '[...]' in the 'new' expression) you ***MUST*** use the '[ ]' in the 'delete' statement. The fact that there is no syntactic difference between a ptr to a thing and a ptr to an array of things is an artifact we inherited from C. ➸

A Concise Guide to UNIX Books

Compiled by: Samuel Ko (kko@sfu.ca, sko@wimsey.bc.ca)

Submitted by Andrew Trauzzi

This month we look at some shell programming books, and UNIX editors.

The Unix C Shell Field Guide
Gail Anderson and Paul Anderson
1986 ISBN: 0-13-937468-X
• The C-Shell Bible - everything you need to know to understand csh and use Unix effectively.

Unix C Shell - Desk Reference
Martin Arick
1992 ISBN: 0-89435-328-4
• A more recent text on maximizing the use of C-Shell.

Unix Shell Programming
Stephen Kochan and Patrick Wood
1990 ISBN: 0-672-48448-X
• Highly Recommended. A classic on using and programming Bourne Shell (and Korn Shell).

Unix Shell Programming
Lowell Arthur
2nd ed. 1990 ISBN: 0-471-51821-2
• This covers not only common shells but also general software tool concepts.

The Korn Shell Command and Programming Language
Morris Bolsky and David Korn
1989 ISBN: 0-13-516972-0
• The authoritative reference.

Unix Desktop Guide to the Korn Shell
John Valley
1992 ISBN: 0-672-48513-3
• This one is easier to read than the work by Korn and Bolsky.

The KornShell User and Programming Manual
Anatole Olczak
1992 ISBN: 0-201-56548-X
• An everything-you-want-to-know-about-KornShell book.

Korn Shell Programming Tutorial
Barry Rosenberg
1991 ISBN: 0-201-56324-X
• A good tutorial on creating Korn shell scripts.

Learning the Korn Shell
Bill Rosenblatt
1993 ISBN: 1-56592-054-6
• Yet another comprehensive text on the Korn Shell.

Unix Editors

GNU EMACS Manual
Richard Stallman
7th ed. 1991
• The official manual of GNU Emacs, essential for emacs users.

Learning GNU Emacs
Debra Cameron and Bill Rosenblatt
1992 ISBN: 0-937175-84-6
• Highly Recommended. Probably the best documentation on editing with GNU Emacs.

Desktop Guide to Emacs
Ralph Roberts and Mark Boyd
1991 ISBN: 0-672-30171-7
• Another good book on emacs.

GNU Emacs Unix Text Editing and Programming
M. Schoonover, J. Bowie and W. Arnold
1992 ISBN: 0-201-56345-2
• Something for everyone who wants to use Emacs.

Learning the vi Editor
Linda Lamb
1990 ISBN: 0-937175-67-6
• A very good guide to vi and ex commands, with a quick reference card.

The Ultimate Guide to the vi and ex Text Editors
Hewlett-Packard
1989 ISBN: 0-8053-4460-8
• Another decent text on vi and ex.

vi Tutor and vi Reference
Robert Colon et. al. (Tut), Maarten Litmaati (Ref)
2.1 (Tut), 8 (Ref)
The latest interactive tutorial (`vitutor2.1.shar(.Z)`) can be obtained by anonymous ftp from `ftp.mines.colorado.edu` (in `/pub/tutorials`). And the refence and other vi stuff are obtainable by anon-ftp from `ftp.uwp.edu` (in `/pub/vi`).

ACCENTServer News

HP, IBM, Novell and SUNSOFT Announce CDE Progress

Common Desktop Environment (CDE) has been delayed awhile to enable more testing with customer, gain consensus and smooth-out some wrinkles in integration with the various vendors' software products.

The good news is that the second snapshot of the CDE effort is available.

In theory, CDE was designed to be an easy-to-use desktop environment which would unite the UNIX offered by various software vendors. In its final form, CDE will allow software developers to create applications that look and behave the same way when run on any UNIX system that incorporates the CDE interfaces.

An updated specification of CDE will be submitted to X/Open in the third quarter of 1994 to facilitate the Fast-Track acceptance process. As part of the ongoing development efforts, HP, Novell, IBM, and SunSoft agreed to extend the early-access phase to accommodate user input and to facilitate smooth system migration.

The availability of the second snapshot and the agreement of the companies to extend the early access phase to accommodate user inputs indicates there is still a strong commitment by the four companies to provide a Common Desktop Environment to the UNIX community.

UNIX Q&A

Originally Compiled by Ted Timar

Submitted by Andrew Trauzzi

Question 1: How do I tell inside .cshrc if I'm a login shell?
When people ask this, they usually mean either: How can I tell if it's an interactive shell? or How can I tell if it's a top-level shell?

You could perhaps determine if your shell truly is a login shell (i.e. is going to source ".login" after it is done with ".cshrc") by fooling around with "ps" and "$$". Login shells generally have names that begin with a '-'. If you're really interested in the other two questions, here's one way you can organize your .cshrc to find out.

```
if (! $?CSHLEVEL) then
  # This is a "top-level" shell,
  # perhaps a login shell, perhaps a shell started up by
  # 'rsh machine some-command'.
  # This is where we should set PATH and anything else we
  # want to apply to every one of our shells.
  setenv   CSHLEVEL    0
  set home = ~username   # just to be sure
  source ~/.env      # environment stuff we always want
else
  # This shell is a child of one of our other shells so
  # we don't need to set the environment variables again.
  set tmp = $CSHLEVEL @ tmp++
  setenv   CSHLEVEL    $tmp
endif
# Exit from .cshrc if not interactive, e.g. under rsh
if (! $?prompt) exit
# Here we could set the prompt or useful aliases
# for interactive shells only.
source ~/.aliases
```

Question 2: How do I construct a shell glob-pattern that matches all files except "." and ".." ?
You'd think this would be easy.

`*`	Matches all files that don't begin with a ".";
`.*`	Matches all files that do begin with a ".", but this includes the special entries "." and "..", which often you don't want;
`.[!.]*`	(Newer shells only; some shells use a "^" instead of the "!"; POSIX shells must accept the "!", but may accept a "^" as well; all portable applications shall not use an unquoted "^" immediately following the "[") Matches all files that begin with a "." and are followed by a non-"."; unfortunately this will miss "..foo";
`.??*`	Matches files that begin with a "." and which are at least 3 characters long. This neatly avoids "." and "..", but also misses ".a" .

So to match all files except "." and ".." safely you have to use 3 patterns (if you don't have filenames like ".a" you can leave out the first):

```
        .[!.]* .??* *
```

Alternatively you could employ an external program or two and use backquote substitution. This is pretty good:

```
        'ls -a | sed -e '/^\.$/d' -e '/^\.\.$/d''
```

(or 'ls -A' in some Unix versions)
but even it will mess up on files with newlines, IFS characters or wildcards in their names.

Question 3: What's wrong with having '.' in your $PATH ?
A bit of background: the PATH environment variable is a list of directories separated by colons. When you type a command name without giving an explicit path (e.g. you type "ls", rather than "/bin/ls") your shell searches each directory in the PATH list in order, looking for an executable file by that name, and the shell will run the first matching program it finds.

One of the directories in the PATH list can be the current directory "." . It is also permissible to use an empty directory name in the PATH list to indicate the current directory. Both of these are equivalent

for csh users:

```
        setenv PATH :/usr/ucb:/bin:/usr/bin
        setenv PATH .:/usr/ucb:/bin:/usr/bin
```

for sh or ksh users

```
     PATH=:/usr/ucb:/bin:/usr/bin export PATH
     PATH=.:/usr/ucb:/bin:/usr/bin export PATH
```

Having "." somewhere in the PATH is convenient — you can type "a.out" instead of "./a.out" to run programs in the current directory. But there's a catch.

Consider what happens in the case where "." is the first entry in the PATH. Suppose your current directory is a publically-writable one, such as "/tmp". If there just happens to be a program named "/tmp/ls" left there by some other user, and you type "ls" (intending, of course, to run the normal "/bin/ls" program), your shell will instead run "./ls", the other user's program. Needless to say, the results of running an unknown program like this might surprise you.
It's slightly better to have "." at the end of the PATH:

```
        setenv PATH /usr/ucb:/bin:/usr/bin:.
```

Now if you're in /tmp and you type "ls", the shell will search /usr/ucb, /bin and /usr/bin for a program named "ls" before it gets around to looking in ".", and there is less risk of inadvertently running some other user's "ls" program. This isn't 100% secure though - if you're a clumsy typist and some day type "sl -l" instead of "ls -l", you run the risk of running "./sl", if there is one. Some "clever" programmer could anticipate common typing mistakes and leave programs by those names scattered throughout public directories. Beware.

Many seasoned Unix users get by just fine without having "." in the PATH at all:

```
        setenv PATH /usr/ucb:/bin:/usr/bin
```

If you do this, you'll need to type "./program" instead of "program" to run programs in the current directory, but the increase in security is probably worth it.

GNU Review

By Peter Graham

I'm writing this article one whole day before the deadline. (Well, actually just about an hour or so before midnight the day before the deadline.

I also have to do a slight change of plans due to a tight schedule. I'll take about Gnu make next month and cover CVS this month.

CVS - An RCS Front End

What is CVS? Well, we are told in the man page that it stands for Concurrent Versions System. That doesn't tell us much. Certainly not much more than "An RCS Front End". Thus, to pilfer from the README file provided with the distribution (which pilfered from the man page)...

"cvs is a front end to the rcs(1) revision control system which extends the notion of revision control from a collection of files in a single directory to a hierarchical collection of directories consisting of revision controlled files. These directories and files can be combined together to form a software release. cvs provides the functions necessary to manage these software releases and to control the concurrent editing of source files among multiple software developers."

CVS requirements

Since CVS is a front end to RCS, you better have RCS installed on your system before trying to use CVS. :-O (RCS version 5.6 or later is preferred). CVS also uses NDBM. It includes its own code for cross-platform support which is adequate for all but the largest applications. If you feel you will use CVS heavily, it is better to use your native NDBM. This can be done by editting 'src/config.h'. NDBM is probably available on pretty much all systems now.

CVS installation

Unlike, its friend RCS, CVS is pretty much a standard Gnu install. You begin by checking 'src/config.h'. The only thing I had to do there was add the '-a' option to the diff program specification since I use Gnu diff (the preferred option). You must then run the auto-configure script './configure'. Carry on with a 'make' and (as root) 'make install' and then finish things up with a './cvsinit' to initialize the CVS system. For this final step you will need to specify the CVS root repository where all versions of your software project(s) will be maintained. Pick a filesystem with plenty of free space if you are going to make heavy use of CVS. Of course, if you are running CVS in a networked environment (like many of us don't) you should choose a filesystem which is NFS/RFS/AFS/... mounted appropriately.

CVS Components

Unlike RCS, CVS is a single program with many options (a.o.t. many programs with few options). For you graphics people, its sort of the 'bitblt' of revision control. Much of what it does looks like RCS commands (as would be expected). For example, you check in and check out code. Rather than doing this on a per-file basis though you typically do it on a per-module basis. A module may consist of many files organized hierarchically. There is also a 'mkmodules' command which rebuilds the modules database. This is infrequently used.

CVS Usage

The first thing to be done to use CVS is to set the CVSROOT environment variable in your login script to point to the CVS repository specified during './cvsinit'. CVS has many options and sub-commands. You can generally start off (maybe even get by) with a few of them. The general syntax of a cvs command is 'cvs [cvs_options] cvs_cmnd [cmnd_options] [cmnd_args]'. The essential cvs commands ('cvs_cmnd') are:

cvs checkout <modules>
- Checks out the named module(s), each of which consists of a file or hierarchy of files. This command creates a local copy of the file(s) for you in the current directory and also records usage information so that other programmers can be prevented from concurrently updating the same module(s).

cvs update
- When executed within the local copy of the module(s) extracted by 'checkout' this command causes your file copies to be updated with any changes made by other programmers (assuming concurrent updates are permitted).

cvs add <file>
- Adds a new file to the current local module which will be added to the repository on the next 'commit'.

cvs remove <file>
- After deleting the local copy of a file, this command can be used to cause the eventual removal of the file from the module in the repository (again, at 'commit' time).

cvs commit <file>
- This command causes your changes to be incorporated into the module in the repository.

cvs import <file>
- Allows new modules to be added to the repository. The <file> specified will usually be a directory containing the files composing the initial implementation of a project.

There are many more CVS commands and also many options associated with them. They are far too numerous to discuss here. See the man page for more information.

CVS Summary

Name	CVS (Concurrent Versions System)
Description	Front end to RCS allowing hierarchies of files to be managed with version control.
Archive Loc'n	prep.ai.mit.edu: /pub/gnu/cvs-1.3.tar.gz
Archive size	419164 bytes
Approx Install Space	3MB
Install Time(Sparc-1)	A little under 10 minutes.
Pros	• Free, small, and easy to install. • Supports hierarchically structured projects. (This is REALLY important for large projects.) • Specific mailing list in support of CVS. `<info-cvs-request@prep.ai.mit.edu>` for requests to be added/deleted. `info-cvs@prep.ai.mit.edu` for bug reports, questions and the like.
Cons	• Storing all projects on one file system is a little restrictive if you are tight on space. Symlinks are, of course, an answer.

See you all at the next meeting. Anybody else looking forward to the summer BBQ meeting? Requests, as always, to `<pgraham@cs.umanitoba.ca>`.

The PowerOpen White Paper

Part 1 — The PowerPC Architecture

Submitted by Keri Gustafson through Bary Finch

The PowerPC alliance has the "Power" to change desktop computing as we know it today. In this month's and next month's newsletter, the original IBM white papers will be presented in full. This month's paper will discuss the PowerPC architecture and the impact it will have on future computers. Next month's paper will discuss the PowerOpen consortium and what it means to the computing community. Both papers have been submitted courtesy of IBM. (See the end of this article for copyright information.)

The PowerPC(TM) Architecture

The PowerPC Architecture, introduced with the Apple, IBM, and Motorola alliance in October 1991, has seen wide acceptance by major system vendors. These vendors have adopted this common RISC architecture to span all types of computing platforms from laptops to supercomputers with software compatibility throughout. PowerPC microprocessors will also become the core for a variety of embedded controllers for such applications as automobiles and other real-time, mission-critical products.

The PowerPC Architecture is based on the existing POWER™ (Performance Optimization With Enhanced RISC) architecture used in various system product lines from multiple vendors originating with IBM's successful RS/6000 line of workstations and servers. This architectural heritage will allow PowerPC products to take advantage of the large installed base of POWER software applications already in place for those platforms. The name "PowerPC" reflects the refocus of the architecture in a form suitable for very high volume, single-chip microprocessors. This, combined with its new multiprocessor architectural features, will make PowerPC processors ideal solutions for systems ranging from low cost portable and desktop computers all the way up to symmetrical multiprocessors and highly parallel supercomputers.

History Of RISC Technology

The basic ideas behind Reduced Instruction Set Computers (RISC) were developed in the mid 1970s at IBM's T.J. Watson Research Center by John Cocke, and embodied in a machine called the IBM 801 minicomputer [Radin82]. These ideas were further refined and popularized by a group at the University of California in Berkeley led by David Patterson, who coined the term "RISC" [Paterson81]. These early RISC pioneers realized that the then prevalent trend toward more complex instruction set computers (embraced by "CISC" processors such as the VAX, 8086, 32000, and 68000 processor architectures) was not the best approach for building future high performance processors.

The primary motivation for more complex instruction sets, up to that time, had been the desire to reduce the "semantic gap" between the instructions executed by the processor and the high-level languages in which people were programming. This notion was based on the intuitively appealing theory that such a processor would have to execute fewer instructions (have a shorter path length) and would, therefore, naturally have better performance. The key observation made by early RISC researchers, however, was that the existence of a microcode interpreter in the processor to execute these complex instructions introduced an expensive overhead that actually slowed down execution of the more frequently occurring simple instructions — with a net loss in performance. Furthermore, complex instructions proved to be a rather poor target for compilers because it was difficult to make effective use of them and in many cases they precluded optimizing away unnecessary operations.

With larger, faster, less expensive memory devices and improved compiler technology available, it became feasible to consider simplifying the instruction set, even with the potential cost of larger code size and higher memory bandwidth requirements. The 801 was the first machine to implement this strategy. It successfully demonstrated that simplifying the instruction set enabled implementations with smoother running (bubble free) pipelines that approached the goal of single-cycle instruction throughput. It also showed that investing more transistors in instruction throughput and fast cycle times, produced a more optimal solution to the computer performance equation than was possible by spending those transistors on complex instructions.

These basic RISC ideas turned out to be even more significant than originally thought. Not only did RISC processors demonstrate more parallelism through better pipelining, they also made the idea of dispatching multiple instructions simultaneously (superscalar1) tractable. This is the essence of RISC architecture. It allows the execution of more operations in parallel and at a higher rate than is possible with a CISC architecture using similar implementation complexity.

Satisfied that the 801 concepts had made significant improvements in instruction cycle times and pipeline efficiency, IBM set out to improve on the 801 architecture by:

1) explicitly embodying the concept of superscalar operation in the architecture;
2) improving the architecture as a target for compilers;
3) further reducing instruction path lengths; and
4) including floating-point in the architecture.

This effort culminated in the development of the POWER™ Architecture [Oehler90] in the late 1980s, which now forms the basis of IBM's RISC System/6000T family of workstations and servers.

In The Beginning, There Was the POWER Architecture

The POWER Architecture is a conventional RISC architecture in most respects; it adheres to the most important of the RISC tenants:

1) Fixed length, consistently encoded instructions.
2) A register-to-register (load/store) architecture with ☛

primitive addressing modes.

3) Relatively "simple" instructions.

4) A large, orthogonal register file.

5) Three operand (non-destructive) instruction format.

However, the POWER Architecture also has several features that set it apart from other RISC architectures:

First, it was organized around the idea of superscalar instruction dispatch. Conceptually, instructions are dispatched across three independent execution units, a branch unit, a fixed-point unit, and a floating-point unit. Instructions can be dispatched to each of these units simultaneously, where they can execute concurrently and finish out of order. Execution units adjust to the dynamic instruction mix by "slipping" past each other. To facilitate this, each of the conceptualized execution units has an independent set of resources to minimize communication and interaction between units. And, while the execution units can complete instructions out of order, the units are synchronized transparently to the software by fully interlocked instruction pipelines. This not only simplifies programming, it also assists in assuring software compatibility across multiple different implementations.

Second, the POWER Architecture included several "compound" instructions to reduce the instruction path length. Perhaps the only drawback to RISC technology vis-a-vis CISC technology, is that it takes more instructions to perform a given task. But IBM recognized that much of this code expansion is avoidable with minor enhancements to the instruction, which do not constitute a return to full blown complex instructions a la CISC. For example, a large fraction of the code expansion was found to be due to the prolog and epilog code associated with saving and restoring registers across a procedure call. To eliminate this factor, IBM introduced "load and store multiple" instructions that allow several registers to be moved to or from memory with a single instruction. Another example is the automatic update of the base address register on loads and stores, which eliminates extra instructions to increment the index when striding through arrays. Even though this is a compound operation, it does not adversely effect the RISC pipeline flow because the updated address is already available and a register file port is normally available while waiting on the memory operation. Other path length reducing features include such things as:

1) an extensive set of bit-field manipulation instructions;
2) compound multiply-add floating-point instructions;
3) condition register setting as a side-effect of normal instruction execution; and
4) load and store string instructions (which load or store arbitrarily aligned).

A third factor that differentiates the POWER Architecture, is the absence of the branch-and-execute capability found in the 801 and many other contemporary RISC machines. Branch-and-execute (sometimes called delayed branching) causes the instruction following a branch to execute before the branch gets taken. This feature worked effectively in early RISC machines to fill the instruction bubble created by branch evaluation and fetching the new instruction stream. However, in more advanced, superscalar machines, this feature is both ineffectual and burdensome. It is ineffective because a single cycle branch delay induces multiple instruction bubbles that cannot all be covered with a single architectural delay slot. Such machines will nearly all implement much more exotic facilities (e.g., branch target caches) for covering these bubbles, which render the delayed branch useless. Furthermore, delayed branching is a burden in machines that perform speculative execution past branches because the delayed branch introduces significant complexity in the instruction sequencing logic. As a result, the delayed branch was not included in the POWER Architecture.

"Floating-point is not an afterthought or optional add-on with a clumsy coprocessor interface"

The branching technique used in the POWER Architecture is a fourth unique feature of the architecture compared to other RISC processors. The POWER Architecture uses an enhanced condition register facility. The problem with traditional condition register architectures, is that they pose a performance limitation in the two ways: first, setting a condition code as a side-effect of instruction execution limits a compiler's opportunity to rearrange code, and second, a condition register is a single architectural resource that causes a bottleneck in a machine that executes multiple instructions in parallel or out of order. Some RISC architectures (e.g., MIPS and the 88000) avoided the problem by eliminating the condition register and requiring conditions to be explicitly set (by a compare instruction) in a general register and providing a set of conditional branch instructions that test a general register on the fly (e.g., branch on zero).

The POWER Architecture, on the other hand, fixes the problems of the traditional condition register approach by:

1) providing an opcode bit in each instruction to make the condition register update optional thereby restoring the compiler's ability to rearrange code, and
2) providing multiple condition registers (eight) to produce a large condition register namespace and thus avoid the single resource problem.

This approach supports the conceptual organization of the machine into independent execution units. Conceptually, the condition register is contained within the branch unit along with the branch address registers. As a result, it is not necessary to access the general register file (in the fixed-point unit) to evaluate and execute a conditional branch. This organization leads to the concept of "zero-cycle branching." In this concept, to the extent the compiler can schedule setting the condition code and loading the branch address registers early, the hardware can lookahead and remove, or fold out, resolved branches from the instruction stream. This avoids the instruction issue slot normally required by the branch instruction and allows a continuous linear stream of instructions to flow to the fixed and floating-point units.

A fifth aspect of the POWER Architecture different from some other RISC architectures, is that it embraces floating-point as a first class data type. Floating-point is not an afterthought or optional add-on with a clumsy coprocessor interface. It is directly supported in the instruction set architecture just like standard integer and logical data types. This makes it more well integrated into the overall scheme and encourages much higher floating point performance. This recognizes the increasing importance of floating-point in a wide range of application domains. The architecture supports the IEEE-754 standard floating-point format. It provides a set of 32, double-precision floating-point registers in the floating-point unit that are separate from the general registers in the fixed-point unit. The architecture supports both single- and double-precision data, but single-precision values in memory are converted to double-precision format when loaded from memory to registers, and all arithmetic instructions operate on double-precision data. The floating-point instruction set includes a set of multiply-and-add instructions that can dramatically improve the performance of many algorithms.

PowerPC Architecture: The Vision, The Solution:

Satisfying the diverse needs of the three originator companies and meeting their combined long term vision of computing, required some modifications to the POWER architecture. So, with the goal of maintaining RISC System/6000 software compatibility, a team of architects from IBM, Apple, and Motorola set out to refine the architecture. For example, the "rich" POWER instruction set was pared back to better facilitate low-cost, single-chip versions. Also, a few features were removed to simplify construction of future very aggressive superscalar implementations. And, to satisfy the demands of future software systems, the architecture was extended to a full 64-bits with strong multiprocessor support. These changes resulted in a new architecture, officially called the PowerPC Architecture, which will form the basis for next generation products from many companies.

"On a 64-bit implementation, the machine uses the full 64-bit logical address space"

The PowerPC Architecture maintains the same basic programming model and instruction opcode assignments as the POWER Architecture. Care was taken so that the POWER architecture features removed from PowerPC architecture can still be tapped and emulated to permit PowerPC processors to run existing POWER binaries. The most significant differences between the POWER and PowerPC architectures include:

1) elimination of the MQ register and all extended precision shifts and integer multiply and divide instructions which use it (to facilitate multi-issue superscalar implementations);
2) elimination of four instructions whose operation was dependent on source operand value (to reduce cycle time);
3) elimination of several bit-field instructions that had three source operands (to avoid the need for an extra general register file port);
4) elimination of support for the rarely used corner cases of several instructions (to simplify implementation);
5) elimination of the "load-string-and-compare-byte" instruction which was the most complex instruction in the POWER Architecture (to simplify implementation);
6) addition of unsigned integer multiply and divide;
7) addition of a fixed-point subtract which does not update carry;
8) addition of single-precision floating-point instructions (POWER Architecture only supported double precision which precluded implementations with fast single and slower double-precision);
9) provision for a fast-trap-and-emulate mechanism for implementing complex operations such as string instructions (for low cost implementations);
10) an improved set of instructions for explicitly scheduling data into and out of the cache under user control;
11) definition of a weak storage ordering model (to simplify dynamic reordering of memory operations in hardware) with user storage locking and synchronization (for multiprocessors);
12) addition of a little-endian addressing mode switch;

and last, but not least,

13) extension of the architecture to a true 64-bit model with full support for 64-bit integers and address pointers while maintaining complete compatibility with 32-bit applications.

The most significant change to the architecture was the extension to 64 bits. The architecture allows both 32- and 64-bit versions of the PowerPC processors, but all processors will run 32-bit applications as a minimum. 64-bit implementations will have a 32/64-bit mode switch selectable from supervisor code. 32-bit applications can run on 64-bit implementations with a 64-bit operating system kernel. The extension simply increases the size the registers to 64-bits and adds a few new instructions for 64-bit operations, like 64-bit shifts, compares, and double-word fixed-point loads and stores. The 32/64-bit mode switch actually has very little effect on the operation of the hardware and therefore doesn't have the undesirable effects often associated with mode switches. Nearly all instructions are mode independent, the mode switch merely controls how much of a 64-bit effective address is translated to a physical memory address. On a 64-bit implementation, the machine uses the full 64-bit logical address space, which gets translated to an even larger virtual address space, and then finally translated to a somewhat smaller (but still enormous) physical memory address. The address translation mechanism is consistent and compatible with the translation mechanism used on 32-bit implementations. ☛

The mode switch effects only a couple of other minor aspects of the hardware such as selecting how overflow and carry conditions get set and how many count-register bits are tested on a conditional branch.

PowerPC Silicon For All

The first microprocessor based on the PowerPC architecture, the PowerPC 601™ microprocessor, is now being sampled by IBM and Motorola. It is a medium-sized, medium-performance processor suitable for low- to medium-cost desktop systems. This device also supports the enhanced multiprocessor features for high-end system design. This first device was based on an existing IBM single-chip processor, but has had major enhancements to improve performance and reduce costs. For example, the part has a more sophisticated branch unit and implements the Motorola 88110 high-performance multiprocessor bus interface [Gullette92]. It is a superscalar design capable of dispatching three instructions per clock, with a large, 32KByte, 8-way set-associative on-chip cache. The part is fabricated in an advanced 4-level metal, 0.6um, 3.6volt CMOS process with 2.8 million transistors on a 11x11mm die, and has an estimated performance of over 50 SPECint89 and 80 SPECfp89 at 66MHz.

IBM and Motorola, with Apple engineering participation, have put into operation a new design center to develop future PowerPC microprocessors. The Somerset design center is a 37,000 square-foot facility located in Austin, Texas, staffed primarily by Motorola and IBM with approximately 300 engineering professionals. The design center is currently working in parallel on three separate PowerPC microprocessors. The three parts currently in development in the design center include:

1) The 603: a processor intended primarily for the cost sensitive, desktop and portable personal computer systems, such as those in which Apple might use the 68030 today.
2) The 604: a high performance part for uniprocessor or multiprocessor desktop personal computers and workstations.
3) The 620: a 64-bit high-performance part for high-end workstations, servers, and multiprocessor systems.

Engineers in the new design center are employing a formal VLSI design methodology derived from the best of both IBM's and Motorola's CAD tools. These tools will combine the rapid design capability of the IBM tools with the dense packing capability that Motorola has used to produce very high volume, high yield microprocessors for the commercial market. The new designs will use an advanced 0.5μm semiconductor technology using a common set of design rules for both IBM and Motorola fabrication facilities. Motorola will produce these parts in very high volume in their new MOS11, sub-micron wafer fab in Austin, Texas. IBM will manufacture these microprocessors in their advanced semiconductor facility in Burlington, Vermont. IBM will also sell these microprocessors in the OEM market.

In addition to the four processors described above, new PowerPC processor implementations are in development at Motorola internal design centers. These designs are targeting specifically at the high volume, very low cost embedded control markets as well as the low power, sub-notebook computer markets. Also, research is underway into advanced microarchitectural techniques for the next generation of billion-instruction-per-second class microprocessors to lead the way into the 21st century.

Summary

The PowerPC Architecture represents the culmination of nearly 20 years of work on RISC architectures beginning with IBM's seminal work in the 1970s and refined both by Apple's experience in advanced personal computers and by Motorola's experience in delivering low-cost single-chip microprocessors into high-volume markets. With their combined resources, the originator companies intend to deliver the broadest range of RISC microprocessors available in the industry. The PowerPC architecture will drive the power of RISC down to the very lowest end of the portable computer marketplace as well as the highest end of the supercomputer market.

— Footnotes —

1) The term "superscalar" is believed to have been coined by T. Agerwala and John Cocke [Agerwala87]. It refers to the machines capable of dispatching multiple instructions per clock from a conventional linear instruction stream.
2) Of course, any given processor may implement each of the three conceptual units as multiple execution units to allow more than three instruction per clock issue.
3) The linkage conventions used by the POWER compilers are very powerful, satisfying in one simple, unified mechanism the problems of relocation, shared libraries, and dynamic linkage. This is done by indirect addressing through a table-of-contents (TOC) which is updated at load time. The load and store multiple instructions are critical to the linkage conventions used in POWER software.

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

(Next month's column will present the IBM White Paper on the PowerOpen association.)

SIG Sideline

By Brad West, SIG Coordinator

There was no specfic topic this meeting, so the round table format was followed. The evening discussions centered mainly around Linux with topics ranging from what tape drives are supported, to how stable is Linux alongside other operating systems on the same hard drive, to how well sound cards work. The Slackware distribution is at version 1.2.0 and the kernel is at version 1.0.8. It was noted at the meeting that the full distribution of Slackware for Linux version 1.2.0 is available at the U of M on ftp site <`ftp.cc.umanitoba.ca`> in directory "`/SLS/slackware`". Also if anyone is interested in borrowing the Slackware distibution set of diskettes, you can contact Greg Moeller <`gkm@muug.mb.ca`>, or come to the next SIG meeting.

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 at <`bwest@muug.mb.ca`> or my work phone is 983-0336. There is no specfic topic for next month's meeting and in the event that a topic is not found, the round table format will be followed. The next meeting is scheduled for Tuesday, May 17, 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. We'll see you at the May meeting.

Accent Server News

Don't leave your node without it!

(The Free Suranet Guide to Internet Resources, that is)

The latest version of SURAnet's Guide to selected resources on the Internet is available for FREE from a couple of different sources. We highly recommend this document. It is kept up to date and is user-friendly for neophytes. Here's how to get it:

1. Send e-mail to `flash@admin.sun.com` with a subject of "63.02". (Note: comes in two pieces — a total of 140K bytes.)
2. Anonymous ftp to: `ftp.sura.net:/pub/nic/infoguide.XX-XX.txt` (Note: where XX-XX denotes the latest publication date such as 03-93).
3. Send e-mail to `accentserver@nis.com` with the subject of "SURANET GUIDE". (Note: comes in seven pieces — a total of 140K bytes.)

This Month's Speaker

This month, Rob Dempsey will present "Trust in a Distributed Computing Environment". Issues discussed will be:

- Understanding of the issues facing administrators in a DCE.
- Understand technologies which can be used to address these issues, specifically the Kerberos Authentication Service.
- Understand issues you will face in a successful implementation of these technologies.

Rob is an Open Systems consultant with the HP Professional Services Organization. His focus is on the security of systems in the client-server environment.

Rob has worked in many corporations in Canada and abroad, including Great-West Life, IPL, Canadian Utilities, ScotiaBank and CitiCorp. He is currently co-authoring a book on distributed computing security.

Rob has been a public speaker, instructor and author of several articles in the technical computing fields. He is a professional accountant (CGA), and holds a B.Sc. from Sir George Williams University in Montréal. Rob resides in Calgary, Canada.

Agenda

for
Tuesday, May 10, 1994, 7:30 PM
Samuel N. Cohen Auditorium
St-Boniface Hospital Research Centre
Main Floor, 351 Taché

1.	President's Welcome	7:30
3.	Business Meeting a) Old Business b) New Business	7:35
5.	Presented Topic "Trust in a Distributed Computing Environment" Presented by Rob Dempsey of HP Canada. (See writeup below)	7:45
4.	Coffee Break and Informal Discussion	9:00

Note: Please try to arrive at the meeting between 7:15 and 7:30, to avoid disrupting the meeting in progress.

Coming Up

Meeting:

June's meeting is scheduled for Tuesday, June 10, at 7:30 PM. Meeting location will be the St-Boniface Research Centre, as usual. The June meeting is the annual MUUG barbecue! Stay tuned for details.

Got any ideas for meeting topics? Any particular speaker, company, or product you'd like to see at one of our meetings? Just let our new meeting coordinator, Roland Schneider, know. You can e-mail him at <`rsch@muug.mb.ca`>.

Newsletter:

If you are interested in a particular topic, let me know. I'm sure I could coerce you into writing an article! I could use a few articles — especially shorter ones — half a page to one page (400 to 1000 words) would be fine.

Monsieur Ex has also let me know that his mail-box has room for more of your wonderful queries again – please submit your questions to the old guy via e-mail to <`m-ex@muug.mb.ca`>. He may be old, but he's not ready for retirement yet!