Technical UNIX®User Group

October 1990 $2.50
Volume 3, Number 1

newsletter of the Technical UNIX® User Group

This month ...

Late Breaking News...
Next Meeting to be held at UNISYS
Time to RENEW Your Membership

UNIX is a registered trademark of AT&T.

Thoughts From The Editor

By Susan Zuk

This is the last newsletter for the 1989-90 TUUG year. I was quite excited when I received articles for the newsletter. I had an overflow of information. Thank you to those members sending information and keep it coming. Most of the submissions should be included next month.

It has been discussed that we circulate a full newsletter every other month with a meeting notice in between. If you have any opinions please let us know.

The elections for the upcoming year are scheduled for the October meeting. Please think about becoming involved with the executive or offering any type of helping hand.

October is also the month for renewing memberships. Please present your membership fee ($20.00) at the next meeting or send it to: Technical UNIX User Group, P.O. Box 130, Saint-Boniface, Manitoba, R2H 3B4. We will send a receipt back to you.

This month's newsletter features an article on the multitude of programming features available on UNIX. Take a look to see if you can pick up any tips. Gilbert submitted his final President's column as he is stepping down this month. Thank you Gilbert for all your submissions. Gilbert will continue helping the group in the position of Past President. Take note of the employment opportunity as listed on page 7. It is really nice to see UNIX positions being listed in our city. It has taken some time but the number of UNIX installations is growing!

Hope to see you at the October meeting and I'm ready to receive more articles. Remember to think about becoming not only a member but an ACTIVE member.

Group Information

The Technical Unix User Group meets at 7:30 pm the second Tuesday of every month, except July and August. The newsletter is mailed to all paid up members 1 week prior to the meeting. Membership dues are $20 annually and are due at the October meeting. Membership dues are accepted by mail and dues for new members will be pro-rated accordingly.

The Executive

President:	Gilbert Detillieux	261-9146
Vice President:	Derek Hay	943-5401
Treasurer:	Gilles Detillieux	261-9146
Secretary:	Vacant	
Newsletter Editor:	Susan Zuk	(W) 788-7312
Membership Sec.:	Gilles Detillieux	(W) 788-6209
Information:	Gilbert Detillieux	261-9146
	(or) Susan Zuk	(W) 788-7312

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

Copyright Policy and Disclaimer

This newsletter is ©opyrighted by the Technical UNIX User Group. Articles may be reprinted without permission as long as the original author and the Technical UNIX User Group are given credit.

The Technical UNIX User Group, the editor, and contributors of this newsletter do not assume any liability for any damages that may occur as a result of information published in this newsletter.

ANNOUNCEMENT...

Meeting Location:

The October meeting location will be provided by UNISYS Canada Inc., Suite 1000-1661 Portage Ave (UNISYS Building). Upon entering the building you will then be required to sign-in. Please sign-in using "TUUG" as the agency represented.

President's Corner

President's Parting Shots

by Gilbert Detillieux, President

A while ago, I read in some Unix-related publication a criticism of the Open Software Foundation that went something like this: "The only thing open about them is their mouths." This statement does sound rather extreme, and is an emotional rather than intellectual response, but it is typical of the mud-slinging and rhetoric that has been exchanged between the OSF and the rival Unix International. Seeing this display, I can't help but draw comparisons to politics, and wonder: While these people are looking after their own interests, who is looking after the interests of the little people — the users?

The GUI battle is a case in point. The OSF has just released Motif 1.1, which is based on X11 Release 4 (the original Motif was based on X11R3). The problem is that the new Motif is not binary compatible with the previous version, requiring "a simple recompile" of all Motif client applications. While this slight incompatibility is not likely to pose a problem for any Motif software developers, it could be a problem for the users.

Why is the new version of Motif incompatible? It turns out that Motif was implemented using a modified version of MIT's Xt library (the X Toolkit Intrinsics) from X11R3. To move up to X11R4, the OSF had to incorporate similar modifications to MIT's significantly altered Xt library. The OSF has now made these modifications available to MIT for inclusion in the eventual X11R5, so that should be the end of that problem. This seems like a strange way to be developing an "open" standard — by requiring modifications to other previously defined standards.

Sun's Open Look, which is based on X11R4 doesn't suffer this problem, since it uses its own libraries that interface to the standard X11 library. However, Sun shouldn't gloat too much about the Motif incompatibility problem — the implementation of the Open Look 2.0 libraries was different enough from the previous version that relinking to the new libraries was required.

What all this says to me is that these are still emerging technologies, without all the bugs ironed out. Furthermore, users haven't had a say in any of this yet. Standards are being imposed prematurely by companies looking after their own best interest, without much consideration for the rest of us. Although I am a big proponent of standards for developed technology, I feel that these organizations are imposing standards without foresight, and this is likely to stifle progress in these new areas.

Is it any wonder the boys from Bell Labs have decided to throw out Unix and its whole standards-laden baggage and gone back to the drawing board? Their new project, called "Plan 9," is a highly distributed Unix-like operating system, which they proudly claim does not adhere to any of the present standards. This move is not so surprising when you consider that Unix was originally created by this anarchist group as an escape from the shackles of mainframe-based system standards of the time.

As I now step down as president of the group, I leave you to ponder the future of Unix, and the future of our group. Perhaps the answer to both requires a spirit of cooperation and involvement from all users?

I would like to thank all of you who supported and assisted me during the past two years as president. This has been a very rewarding and interesting experience. I wish the new president, whoever that may be, all the best for the upcoming year. I encourage you all to come out to the next meeting to vote and/or acclaim the new executive, and encourage you all to consider what your role will be in the group's future. We will be meeting at Unisys (10th floor,1661 Portage) at 7:30PM, October 9. Hope to see you there.

The Fortune File

This month's fortune, submitted by Gilbert Detillieux, was produced by the DOS MURPHY program.

All general statements are false.

A Menagerie for Programmers

By Dave Taylor

Reprinted with permission from the October issue of CommUNIXations, published by Uniforun

UNIX systems offer many useful utilities. Here's a collection of fauna native to the world of programming.

A veritable animal kingdom of utilities exists for the UNIX platform. On almost any given machine are hundreds of different commands that you can invoke at any time. What's more, almost all of them have a variety of different command options -- *flags* -- that can completely change their behaviour. For example, using the right flags with the *who* command on System V, you can have it tell you what "zombie" processes (processes not attached to a terminal) are running on the machine. A tour of standard utilities in the programmer's environment and a sampling of commercial products that supplement the basic tools may suggest some useful ways of interacting with any UNIX system.

EDITING

Many programmers spend the most time in the editor, adding new code, reformulating existing code, puzzling out incorrect behaviours or adding comments to enhance the readability of the software. Given this necessity, it is surprising that so little has been done to create sophisticated editing environments that are knowledgeable of programming languages.

Standard UNIX utilities typically are designed to function reliably on virtually any terminal. As a result, UNIX has minimal facilities to aid in the intelligent display and editing of programs, which are difficult to achieve across scores of available terminals. There are various add-ons and options within both *vi* and *EMACS*, but programmers accustomed to the more focused environment of a proprietary system may find them lacking. For example, Symantec's Lightspeed Pascal on the Macintosh has a multifont, Pascal-knowledgeable editor that puts all keywords in boldface as you type them in, to ensure a consistent format for what you're typing. This is especially helpful when you go back to the code and try to remember how the program should work.

For printing source code, however, various programs can help produce more readable formatting. Most notable is *vgrind* from Berkeley UNIX, which translates C source code into a *troff* document that can then be printed on a high-quality printer such as a Versatec. *Tgrind* offers similar functionality, with output in Tex format. Other utilities translate C source into well-organized PostScript.

COMPILING

Once your code appears correct, the next step is to compile it. However, one of the greatest problems with the C programming language is that most compilers are designed simply to translate the source into relocatable object code (which the loader *ld* then uses as input to create an actual executable file). If you have a program that invokes a routine as "testme(1)" and later defines the routine to have five parameters, the compiler will accept that without incident and generate object code. Of course, when you actually execute the program it's likely that things will fail.

What's missing in the traditional UNIX C compiler is a sophisticated syntax and semantics checker. (Indeed, one can argue that the very design of C prohibits a sophisticated check of code.) However, it is true that a number of useful utilities can go a long way to the quest to create correct programs.

The most important of those is surely the *lint* program, an application that accepts a single- or multiple-file C program and tries to verify correct usage, as well as adding the basic semantic checking that the compiler skips. *Lint* would flag an error in the case cited above, in which a routine is invoked with the wrong number of arguments. However, *lint* also teaches programmers about the importance of return types in library calls and the value of the "(void)" cast, which in practice many people ignore. To any program it is given, *lint* is likely to respond with dozens of complaints about incorrect return types, because the programmer has ignored the return code of a library routine. Indeed, all the examples in *The C Programming Language* by Brian Kernighan and Dennis Ritchie (Prentice-Hall, 1978) generate errors in *lint* unless modified.

Catching usage errors isn't always enough. Many varieties of *lint* offer the ability to check for code portability as well. In fact, there are third-party packages, most notably the ANSI Code Verifier, that examine code with an eye toward a specific coding standard or hardware type. On a machine that offers sufficiently good performance, it might be wise to require that programmers remove major problems in their code before compilation takes place. Packages that test a C compiler for ANSI X3J11 compliance, such as the Ace C Validation Suite, are very different from those that check user-generated code against the same standard and are useful only for companies developing compilers or confirming that a particular vendor indeed complies with the ANSI draft standard.

Another aspect of compilation that is essential on all UNIX computers is the *make* facility. Almost all programming projects end up consisting of more than a single source file, so when files are modified, only the minimum number of recompilations should occur to create a new executable. *Make* figures out that minimum recompilation automatically through the rules defined in the project-specific makefile. *Make* should be a part of all UNIX systems. If it isn't on yours, you can obtain it from your vendor.

One of the most difficult parts of dealing with the *make* utility is creating makefiles. There are a couple of different approaches to this in the UNIX community, including new versions of *make* that require dramatically smaller makefiles, like *imake* and *pmake*, and packages that create makefiles from a set of source files, like *mkmf* and *depends*. Either way, the time spent setting up a project to use the *make* facility will be richly repaid later.

DEBUGGING

Perhaps the greatest bane of programmers, UNIX and otherwise, is the lack of truly useful and intelligent debugging packages. Ideally, of course, one would like a program that executes your own program, then outputs something like "logic flow in routine `make_connection()' is wrong; check worst case size=1." We don't have that level of debugging yet but some utilities within the UNIX environment can aid in the creation and validation of correct, error-free applications.

Most notable among them is, of course, the *lint* program already mentioned. One of the classic problems with UNIX programming is errors of mismatched parameter types or number of parameters in library calls. These can quickly corrupt memory (since variables aren't forced or constrained to their own subspace) and cause many confusing and difficult problems.

At run time, however, most UNIX implementations have a "source-level" debugging system available, too, either *dbx*, *cdb* or *xdb*, depending on which system you're on. These allow programmers to set breakpoints in the C source code and examine the state of memory (and files) at any point in the program. Programmers can even alter the value of different variables without leaving the program. On the downside, since you're working in a different run-time environment, it's quite possible that errors will not occur then but will later when run without the safety net of the debugger. Further, all of these source-level debuggers fail to trace forked child processes (that is, if your program invokes another program, or uses *fork* or *vfork* to spawn processes, you have to wait until they're done to regain control of your program).

Powerful CASE tools are the answer at this point, and one of the best is Saber-C. It offers greatly enhanced run-time error checking (including out-of-bounds array indexing) by executing the code from a C interpreter rather than the actual compiler object code. Also in this line is the Safe C Interpreter.

Another utility worth learning the basics of is the assembly-level debugger *adb*. It lacks the features of the slicker symbolic debuggers but offers programmers the ability to examine a core dump and figure out what routine caused the failure (including a stack trace of which routine called that routine, all the way back to the main program). Its cryptic interface makes it difficult to use but worthwhile to know when you encounter sporadic failures in a package.

RUN-TIME PROFILING

Intimately coupled with being able to debug a running program is the ability to understand what's being invoked and how often routines are called. Not only does this aid in understanding the software, it is also invaluable for performance tuning. Instead of rewriting all the routines to speed them up, it's much more sensible to find those simple little routines that are invoked thousands of times and spend the effort maximizing their performance.

For most UNIX machines, this capability is accessed by compiling the code with the -p compiler option (for run-time profiling). Invoking the code that's been compiled for profiling will create a status log file (typically called "mon.out" or something similar), which you can then examine with the *prof* command. Output of this two-step profile can include subroutines sorted by number of calls or graphical output through use of the plot command. There is also a *gprof* command for different graphical profiling, as well as the *pxp* Pascal execution profiler, which seems to be found exclusively on Berkeley UNIX systems. Some stand-alone packages also offer this power, including Saber-C, the C Dynamic Analyzer and the Safe C Dynamic Profiler.

SOURCE CODE CONTROL

UNIX has always offered top-notch tools in revision or source code control systems. The idea behind these packages is quite straightforward; as software is modified, it's important to keep an audit trail of changes that have been made. This allows projects to back up to a specific date, for example, in order to duplicate exactly the code with which a customer is having problems, as well as to understand how software evolves throughout the lifetime of a large project.

The premier tools for source code control are the Source Code Control System (SCCS), found on almost all UNIX systems, and RCS, developed by Walter Tichy at Purdue University. Both offer similar functionality, with utilities to "check in" and "check out" files from the file store, to annotate changes to indicate what was done and why, and to extract specific versions by number or date. Any programming project that employs more than one programmer should utilize either one of these two or a commercial package that offers similar functionality. These tools can also be useful for tracking modifications to normal text documentation such as technical or procedures manuals and other non-program files.

Commercial packages offer a great deal more functionality than either SCCS or RCS. Among them are Ace CADese, Aide-De-Camp Software Management System, GNJ Configuration Management

System, Cradle, Software Backplane and TeamNet Data and Configuration Management System.

SOURCE CODE TRANSLATION

A bit less common but extremely useful when needed are applications that take as input a program written in one programming language and output it in a different language. These are commonly referred to as source code translators, to distinguish them from compilers that translate source into object code for linkage and execution.

Most of the language translators offer translation into the C programming language from various shell languages and different programming languages, including Pascal, Basic and Fortran. Among the public packages available in this corner are *p2c*, a Pascal-to-C translator, and *ptoc*, an earlier attempt at that. Because these are large, complex packages, it's not surprising that there are many more commercial packages to aid in source translation, including translators for Basic to C, Fortran to C, Pascal to C, Fortran 77 to C++, Cobol to C and the C shell to C.

OTHER USEFUL UTILITIES

Various directions are worth investigating to improve your existing development environment. One of the best things a UNIX software developer can do is learn the ins and outs of the many different applications and tools on the UNIX system, and how they interact. Don't hesitate to write small packages for specific functionality that isn't otherwise offered -- for example, almost every project that uses SCCS has its own custom shell scripts to aid in finding files -- or to customize the interface for your needs.

Specific UNIX tools worth learning more about include *awk*, an interpreted language with string-oriented features and terrific power; *sed*, a stream-oriented editor for performing quick and easy translations on files along a pipe; *lex* and *yacc*, to aid in quick, error-free generation of compiler front ends (though they can be used for more than that); and *grep* and the grep family of file-searching utilities. Even though this richness of choice is characteristic of the UNIX environment, three basic packages remain essential: *make*, *lint* and SCCS or another source code control system. The few hours required to learn how to use them will more than pay for themselves with improved productivity and reliability of code -- and produce happier programmers, too.

DAVE TAYLOR *is president of Intuitive Systems, a consulting firm in Los Altos, CA, which specializes in internationalization, user interface design and software marketing. He is a contributing editor of CommUNIXations.*

Employment Opportunity

Senior Programmer

XCAN GRAIN LTD. requires a senior programmer with 3-5 years experience in a UNIX environment on midrange computers. The technology currently installed includes UNIX V5.3 operating system, ACCELL/UNIFY fourth generation language/database, and C language programming. We are moving towards a TCP/IP ETHERNET network to include personal computers and remote sites. Preference will be given to applicants having skills in these areas. A working knowledge of personal computers would be an additional asset.

XCAN offers a comprehensive employee benefit package and a competitive salary commensurate with experience and training. XCAN GRAIN LTD. is Canada's largest privately owned grain exporter with head offices in Winnipeg as well as offices in Vancouver, London and Tokyo.

Please send your resume to:
Personnel Manager
XCAN GRAIN LTD.
1200-201 Portage Avenue
Winnipeg, Manitoba

Technical UNIX® User Group

AGENDA
for
Tuesday, October 9th, 1990
7:30pm
UNISYS Building
10th Floor-1661 Portage Avenue

1. Round Table	7:30
2. Business Meeting	8:00
a) Elections & Final Executive Reports	
President	
Vice President	
Treasurer	
Membership Secretary	
Secretary	
Newsletter Editor	
Meeting Co-ordinator	
3. Break	8:30
4. Presented Topic	8:40
5. Adjourn	9:30