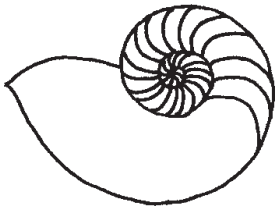
Volume 4, Number 9 September 1992 $2.50



MUUG Lines

Manitoba UNIX® User Group

Newsletter of the Manitoba UNIX® User Group

MUUG Gets a Constitution

By Gilbert Detillieux

As MUUG took a break for the summer, its executive was busier than ever, hammering out a constitution for the group. The ground work for this, jokingly referred to as the Shoal Lake Accord, was done on a rare sunny weekend in July, at the Kwiatkowski cottage. Several revisions and numerous e-mail messages later, we now have a working set of by-laws to present to the group. Once voted in, this will be the first written constitution the group has ever had.

Why all the fuss and bother? First of all, it was felt that the group's objectives should be clearly written out, so that anyone enquiring about the group and its purpose would be provided with a clear and consistent answer. Secondly, as the group grows, and starts to involve more people and more money, it is important to have clear, impartial rules to govern the group's activities and finances. Finally, it is hoped that the set of by-laws we are presenting will result in a more flexible and workable structure for the group's executive, to ensure the continued successful operation of the group.

If you are a member of the group, a draft copy of the by-laws should have been included with your newsletter. We will table this draft for discussion, and possible amendment, at the September meeting, then hold a vote to adopt the by-laws at our annual meeting, in October. In a nutshell, here is what the by-laws are all about, and the changes to the group's structure that they may imply.

Objectives. Essentially, this states the purpose of the group, and is just a statement of what was assumed all along as its objectives. As we are now an affiliate of UniForum Canada, one of our objectives is now to cooperate with this organization as well.

Membership. The big change here is the addition of new classes of membership, such as a (much requested) corporate membership, and provisions for patrons and honorary memberships, as many other groups provide. If your company or organization is interested in corporate membership, read this section for details.

Another change is that membership fees will no longer be pro-rated. Membership will be for one year, effective when you join. Thus membership renewal for new members, after October 1992, will tend to be staggered throughout the year. (Existing members will have to renew this October, and in October of following years, as before.) The reason for this change is to be consistent with UniForum Canada, so that members wishing to join both will have only one renewal date to worry about.

Board of Directors, Executive, Elections. This is perhaps the most fundamental change to the group's structure, but should result in no major impact to members. This structure is intended to give the group's elected officers more flexibility, and adaptability in carrying out their duties. Rather than having you vote for specific individuals for specific executive positions, you will vote for a number of board members. The board will then appoint the executive from its own members. Also, the executive consists only of the required positions.

Additional positions, such as newsletter editor and meeting coordinator, are not defined in the by-laws, as these are better covered as appointments to committees - these can be set up and dissolved by the board at any time, based on demand. Such duties are probably best handled by small committees, rather than a single individual, in any case.

Meetings. Despite the formalism of this section, very little will change to the group's monthly meetings. We will perhaps do away with the "business" portion of the meeting for most months, unless there are any resolutions that need to be passed. The annual meeting will be the October meeting, where elections are held.

It is hoped that these by-laws will ensure the continued smooth functioning of the group, as it grows and evolves. The current draft was made possible by the efforts of a small group of very dedicated members. Now, the next step is up to you, the group members in general. Please look over the draft of the by-laws, and be prepared for the discussion at our next meeting. If you have any questions about them, or require any clarifications, please feel free to contact one of the current executive members by phone or e-mail.

THIS MONTH'S MEETING

Meeting Location:

Our September meeting is scheduled for Tuesday, September 8, at 7:30 PM. The meeting will be held at the U of M, room 234B in the new Engineering Building (near the Senate Chambers). This building is just south of University Centre. Free parking is available in all student lots and most staff lots after 6 PM, except in areas designated as 24 hour reserved.

Meeting Agenda:

See inside for details.

INSIDE THIS ISSUE

Newsletter Editor's Ramblings 2
President's Corner 3
Nominations & Elections 3
Books: So, You Wanna Learn Unix 4
Review: Emacs Text Editor 7
Hands-on: Shared Memory, part 3 8
Announcements:
Cliff Stoll Breakfast Seminar 12
Open Systems Fall Seminar 7
May 12th Meeting Minutes 13
Sept. 8th Meeting Agenda 13

A Newsletter Chock-Full-O' Goodies

By Gilbert Detillieux

Get out the magnifying glass, or update that eyeglass prescription! The September newsletter is full of stuff, and the font sizes are sometimes tiny. (It was either that, or we kill a few more trees.) A lot has happened since the last edition, so there's a lot to cover.

If you're a MUUG member, you should find a copy of the draft by-laws enclosed with the newsletter. The cover article will tell you what it's all about, in case you don't have the time to read it all, or don't want to get bogged down in legalese.

Fall is also election time for the group. Although the election procedures will be different than in the past, they'll still happen in October, with nominations in September. Look for the election committee's report, and list of nominations, later in this edition.

There are also a couple of seminars coming up (one of them is very soon). Look for the seminar announcements for the CIPS breakfast seminar, where Clifford Stoll will be speaking, and the MUUG/CIPS joint seminar on Open Systems, which will be coming up in November.

Also in this issue, look for the article *So, You Wanna Learn About UNIX*, which is reprinted with permission from *Unix/World* (from the July 1992 edition, Vol. IX, No. 7). The article reviews many introductory books on UNIX, and will be quite useful to anyone just starting out, or still unsure of where to look for information about the system. Thanks to Pat Bessler for typing in the article, and thanks to Dave Flack, editor-in-chief at UnixWorld, for giving us permission to copy the article. I've found UnixWorld, in general, to be a very useful source of information on a variety of UNIX-related topics, and I highly recommend it.

Finally, this issue concludes the series on Using Shared Memory, by Peter Graham. The article includes the code for a sample application (this is where the really fine print comes in). We will try to make the source for this (and possibly earlier programming articles) available on-line, possibly via anonymous FTP on the MUUG Online system, in the near future. This will save you lots of typing, if you want to try out the code, not to mention saving us time and paper.

Hope to see you at the September meeting.

The 1991-1992 Executive

President:	Susan Zuk	(W) 788-7312
Past President:	Eric Carsted	1-883-2570
Vice-President:	Richard Kwiatkowski	589-4857
Treasurer:	Rick Horocholyn	(W) 474-4533
Secretary:	Roland Schneider	1-482-5173
Membership Sec.:	Allan Moulding	269-8054
Mailing List:	Gilles Detillieux	489-7016
Meeting Coordinator:	Kathy Norman	(W) 474-8311
Newsletter editor:	Gilbert Detillieux	489-7016
Information:	Susan Zuk	(W) 788-7312
		(FAX) 788-7450
(or)	Gilbert Detillieux	(H) 489-7016
		(FAX) 269-9178

Copyright Policy and Disclaimer

Our Address

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

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. The newsletter is mailed to all paid up members one 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.

PRESIDENT'S CORNER

How We Spent Our Summer Vacation

By Susan Zuk, President

Hi all. I guess with writing this message comes the knowledge that we are quickly approaching the fall season and the completion of another MUUG year! The executive has had a very busy summer working on a number of projects. It is unbelievable the amount of activity we have had these last few months.

As you know, we have kicked off the use of our computer system known as MONA (MUUG Online Network Access). At last count there are 86 users who have the ability to access the system. This has been a real experience and challenge for the executive. There is the extra workload for the group to administer the system as well as the creation of rules, regulations and fee structures which will come into effect on October 1. I would like to especially thank Roland Schneider, Gilles Detillieux, Gilbert Detillieux and Andrew Chan for their continued dedication and hard work in providing the group with such a smooth running system and with the creation of MONA bylaws.

The MUUG executive was also busy creating the Constitution and Bylaws for the group. Since it is such a large undertaking (and I thought that we could whip it together in 6 hours – Hah) we met at Richard Kwiatkowski's cottage and held the *Shoal Lake Accord*. This was a beautiful setting to lay the groundwork for the organization. I can truly say that our meeting was a lot less stressful than our "friends'" constitutional talks. On the final day we had wonderful weather and a chance to take the boats on the lake to do some exploring. A big thanks to Richard for the use of his cottage on that July weekend and for requesting the nice weather. The results of the weekend have been included with the newsletter. It is good for some nice light reading!!! If you have any questions we can discuss them during the September meeting. The Constitution and Bylaws will be voted on at the October meeting.

MUUG's year end is September 30. What this means is that elections are upon us. Nominations are now taking place. Please let the nominating committee know if you are interested in running for a board position (or nominating someone else). The board is elected by the members and then the board members elect the officers of President, Vice-President, Treasurer and Secretary. The other positions are appointed by the President from the board representatives (or possibly from the group membership at large). You will find a current nominations list later in the newsletter. The nominating committee is comprised of Roland Schneider (chairman), Kathy Norman, and Gilbert Detillieux.

Since time does not stand still, I would also like to draw your attention to our fall seminar. This again will be a joint seminar with CIPS and is entitled *Open Systems – Getting Past the Hype*. Please see the write-up in the newsletter for a short overview on the seminar. More information and brochures will be forwarded to you in September.

Just a final note before I close. It is always important for us to receive your input. Please provide us with ideas or requests on meeting topics and also let us know if you are available to volunteer your time. Your comments can be sent to my attention at <zuksue@muug.mb.ca> or please give me a call at 788-7312. Looking forward to seeing you in September.

ELECTIONS

Well, October and the MUUG Annual Meeting are fast approaching. One of the most important events at the annual meeting is the election of a new board of directors. The choice of directors is very important, because they plan group meetings, make policies, raise and spend group money, and generally set the direction for the group as a whole. Almost every event, project, or initiative, from the monthly newsletter to the fall symposium originates with the board of directors. All this can be a lot of work, so it's important to elect people with the energy, enthusiasm, and commitment to do a good job.

You may have noticed that what we once called the "executive" is now the "board." This is in keeping with the new constitution, which will be submitted to a vote at the annual meeting. The only significant difference between the old and the new way of electing the board is that people are no longer voted into specific positions, they are just elected to the board. The board then divides up the positions of president, vice president, secretary, treasurer, newsletter editor, membership secretary, publicity director, and meeting coordinator. This gives the board the flexibility to reassign duties if someone can't continue to do their job part way through the year.

The other change from last year is that we now have an election committee. The election committee does two things: 1) it nominates people for election to the board; and 2) it runs the election and counts the ballots. The list of people nominated by the committee is just a starting point – any voting member of the group can be nominated simply by getting the support of one other member. If you feel you would like to contribute to the group by running for a board position, please don't hesitate to do so. Below is the current list of nominees, followed by instructions on nominating others.

Gilbert Detillieux	Programmer/Analyst	Comp. Sci., UofM
Gilles Detillieux	Programmer/Analyst	Physiology, UofM
Barry Finch	Systems Engineering Rep.	IBM
Paul Hope	Coordinator of A/V and Info. Services	St-Boniface Research Centre
Rick Horocholyn	Sr. Financial Consultant	Manitoba Hydro
Richard Kwaitkowski	Analyst	RK Computer Serv.
Kirk Marat	NMR Supervisor	Chemistry, UofM
Roland Schneider	Graduate student	Elect. Engin., UofM
Rick Thorarinson	Sr. Customer Support Rep.	Unisys
Susan Zuk	Sr. Systems Support Rep.	Unisys

A total of eight people will be elected. If you want to be nominated, or nominate someone else, send a letter to the group's mail-box or deliver it in person to a member of the election committee. The letter must contain the name, title, and employer of the nominee, along with a short (100 word) biography, and must contain the signatures of the nominee and one other member. The letter must be received no later than September 29, 1992. This is 14 days prior to the annual meeting. If you have any questions about the election, please give me, Roland Schneider, a call at 1-482-5173 any time.

MUUG election committee:
Roland Schneider, chairman
Kathy Norman
Gilbert Detillieux

So, You Wanna Learn About UNIX

Because UNIX and easy-to-learn aren't two phrases you normally put together, we compiled a list of books to get you started

By Walter Zintz

Bringing novices up to speed on the UNIX operating system can be a rather daunting task. When UNIX's command line prompt pops up on a terminal screen, the question "Where do I go from here?" pops up in the mind of a befuddled new user. With a software system as powerful as UNIX, It's understandable the beginners might be hard-pressed to quickly master its complexities when armed with only a set of inadequate manuals or poorly organized books.

Although friendlier interfaces are available for most of today's newer systems, they aren't of much help when it's necessary to fix a system problem or do work beyond the ordinary. There are also a significant number of existing UNIX systems without graphical interfaces that require knowledge of such arcane utilities as the **vi** editor, **awk**, and **grep** in order to do the job. If you don't find the UNIX manuals very helpful, there are some well-done how-to books that can help fill the gaps.

There are almost 100 books in print the introduce users to the UNIX system. The majority are for users who have no experience with other computing systems, and hope that they can learn the basics of computing while learning the basics of UNIX. Most of these books are aimed at people who just want to learn to use UNIX for their own work, although there are some good starter books for system administrators and programmers. This article will introduce you to what I regard as the eight best choices for learning both computing and the UNIX system from scratch.

This collection covers UNIX versions from the early, but still much used, Version 7 to the latest System V Release 4 (SVR4), as well as popular variants such as The Santa Cruz Operation's XENIX and UNIX. These books are written for readers ranging from enthusiastic technical types to people whose secret wish is that computers would just go away.

I won't pretend that non-networked systems and character-based terminal screens are the are the future of computing – UNIX systems today feature tight-linked distributed computing and bit-mapped, mouse-controlled, windowed screens. No introductory UNIX book in print today covers such systems, however. For now, beginners on those advanced systems will just have to switch back and forth between any of the following UNIX books and the available network or GUI information sources.

Introducing the UNIX System

By Henry McGilton and Rachel Morgan

For those who will use a UNIX system primarily for its text processing utilities, this is one of the two introductory books – the other being *UNIX for People* – worth your time, and should also be considered if you want to become a proficient user without becoming a nerd. Although this book is based on the older Version 7 of UNIX, introductory UNIX books just don't come any better.

The fact that this book starts from scratch and gives coherent explanations makes it a good choice for even the true beginner. It manages to go beyond the basics to provide extensive coverage of text editing and text formatting, as well as a good start on system administration and UNIX as a programming environment. There's even a solid introduction to shell scripts.

Despite the amount of ground covered, there is never any feeling of being rushed. The authors have the knack of taking just a few words to say something clearly enough that it does not need to be repeated. The book has a lot of information on the ubiquitous **vi** editor and its alter ego, **ex**, as well as the **ed** and **sed** editors. Even with this ambitious list, the authors manage to go into enough detail that readers can become quite expert at online editing. The same holds true for coverage of **nroff**, a cryptic non-WYSIWYG text formatter, the **ms** macro package for taming **nroff**, and the **tbl** preprocessor for creating tables.

There are no nasty surprises when you start to get down to work, either. The authors almost never forget to mention the exceptions to the rules that would otherwise appear to haunt you in real-life use. And while no how-to book is completely free of technical mistakes, this one comes amazingly close – better than any other introductory book I've encountered.

There is one detracting factor: this book is specific to UNIX Version 7, with an extra chapter on some of the Berkeley UNIX extensions. Version 7 is the direct antecedent of almost all UNIX versions presently available, but the systems that use Version 7 or a clone today usually call it something else. Current descendants have new or extended utilities that aren't covered, but the book is still hard to top for learning the basics of UNIX before getting into specific versions.

Don't confuse this book with *Introducing UNIX System V* by the same authors: the books appear similar, but are quite different inside.

The UNIX Operating System

Second Edition

By Kaare Christian

This book is my choice for software developers who are brand new to UNIX, but knowledgeable about computing in general. This group of users will like the book's slant toward UNIX as a programming environment, and the explanations of why various aspects of UNIX are the way they are.

Shell programming and editing in **vi** are the techniques most thoroughly discussed. As for discussion of UNIX internals, it's mostly about the file system, the Bourne shell, and the kernel. There are also chapters on benchmarking – complete with a benchmark shell script – and the relationships between UNIX, PCs, and DOS.

Beyond that, the book focuses on explaining approxi-

mately 100 utilities. But this book is a supplement, not a substitute, for the manuals – it most often restricts itself to peripheral advice on the best way to use a particular utility, and comments on how and why the utility became what it is.

For an ambitious non-UNIX programmer, this book should be ample preparation for *The UNIX Programming Environment* by Kernighan and Pike. The ordinary programmer needn't go that far: this book explains as much about UNIX for programmers as necessary.

UNIX for People

By Peter M. Birns, Patrick B. Brown, and John C. C. Muster

This is the book that will rescue people who don't get along well with computers – it's written for people whose field of expertise is something other than computing. The book's pace is not too fast, no experience in computing is assumed, and the tone is mellow and understanding throughout. Explanations are explicit and thorough, and frequent reminders reinforce what's already been covered.

This book's greatest strength is in its skillful presentation of information. In the course of their varied careers, the three authors got as far as Ph.D. candidacies in education and/or psychology, and they use what they've learned to make a dry, technical subject easier to learn.

On top of that, this is one of the two books I enthusiastically recommend to people who want to specialize in the text-processing end of UNIX use. In this area, the book improves on McGilton and Morgan's book in certain ways, such as dropping coverage of the obsolete **ed** editor in favor of the important **troff** text formatter. As for general UNIX use, it covers most people's needs.

This book's 29 chapters – enhanced with various learning aids, and supplemented by features such as "roadmaps" that demystify certain command transitions – make up a good basic education in UNIX. Readers who want to go farther – without giving up the embedded learning techniques – can contact the authors about the set of follow-on modules they've self-published.

A Practical Guide to UNIX System V

Second Edition

By Mark G. Sobell

One of the strong points of this book is that Sobell is not a UNIX cheerleader – he admits the system has problems as well as benefits. Nonetheless, he had written a fine book for fairly technical readers who want both a text and a reference manual.

Sobell has made significant improvements since the first edition by bringing in topic experts to critique and sometimes rewrite chapters. The result is a well-rounded book.

First and foremost comes a text section that occupies almost two-thirds of the book, and moves the reader pretty far along on everything of importance to users, as well as introduces the programming environment and system administration. Then, distinguished by black page-edge indexing, comes one of Sobell's specialities: reference pages for a large number of commands and utilities, complete with examples. Don't confuse these with the official, cryptic **man** pages provided in UNIX documentation – these lucid pages are more like what the **man** pages should have been.

The book's technical level is rather broad. Thoughtful beginners should have little trouble with the explanations, especially if they pay attention to each chapter's introduction, summary, and exercises. Likewise, people with considerable computing experience on other systems should not find this book slow or pablum.

Peter Norton's Guide to UNIX

By Peter Norton and Harley Hahn

This book is aimed at complete beginners, especially those who are serious about learning both computing and UNIX well. The authors make it clear from the start that using UNIX should be fun. They're right, of course, and none of the other introductory technical UNIX books I've seen encourages this fact.

The writing style is slow, engaging, and helpful. By the time they reach the end of the book, readers will be at the upper apprentice level and well-grounded in the basics. The explanations of the ASCII code are by for the most lucid and complete I've encountered in any beginner's book on UNIX. Sidelights on UNIX history and philosophy are sprinkled throughout, which serves as another tool to keep the book from seeming dry.

The book includes frequent comparisons to DOS, which makes it a good candidate for those who are moving to UNIX from DOS. My only reservation is that the number of technical errors is a little higher than usual among the books on this list.

Using UNIX

By David W. Solomon, Tanya Rodrigue, Mark Schulman, Rosemary Colonna, Dennis Fairclough, and David H. Lender

This is a good choice if your company is only willing to buy one book on UNIX SVR4 for everyone in the company below the level of UNIX guru. It starts out talking to readers who don't know exactly what a computer is, and is filled with clear explanations, screen dumps, and other illustrations.

The book never bogs down at the simplest level: well before the midpoint, the concept of file block sizes as they relate to space wastage and file fragmentation is introduced. The book provides frequent comparisons to DOS, but you need not know that system to follow this book easily.

A reader choosing an introductory book for personal use might also select this book. After the first few chapters, sections can be taken individually, which allows you to dip into it as needed.

UNIX System V Release 4: An Introduction

By Kenneth H. Rosen, Richard R. Rosinski, and James M. Farber

This volume's huge 1,228-page count is devoted to explaining just about everything about SVR4 except the guru stuff. After covering the usual beginner topics in great detail, it goes on to specialized matters like security, network administration, and running UNIX along with DOS. The result is a book with a lot to say to every serious reader, from the complete computer neophyte to the fairly advanced intermediate.

The writing is reasonably clear, although it is a little dry, and written at the right level for determined beginners or ordinary intermediate users. The page allows plenty of room

for extended explanations, sidelights, comparisons to DOS, and reference material. My primary reservation is that the frequency of technical errors is high. It's not any worse than the average introductory UNIX books, but is noticeably below the standard set by the rest of the books in my top eight.

Using SCO UNIX

By Geoffrey T. LeBlond, William B. LeBlond, Sheila R. Blust, Wes Modes, and Ross Oliver

This is the book I recommend to people involved with XENIX and UNIX from The Santa Cruz Operation. It's a very good book on UNIX with a lot of SCO-specific information as a bonus.

This is the only book in my top eight that offers help with installing the system software on a new computer. Another pleasant surprise is the inclusion of chapters on constructing filters, understanding SCO business software, dealing with **uucp**, and running DOS under UNIX.

The technical range is medium, and the writing accessible to fairly astute beginners. After reading the entire book, the reader will be well into intermediate user status. Skimming is pretty safe because the authors use icons in the margin to draw attention to points that shouldn't be glossed over.

Those who like its style, but have other variants of System V, should turn to *Using UNIX System V Release 3*, which is also published by Osborne/McGraw-Hill and has essentially the same structure.

Other Resources

Although there are seminars and video courses galore available, some of which are quite good, they are unnecessary when books like these are available. A judicious selection from among these books should get any beginner off to a great start.

*Editor-at-Large **Walter Zintz** writes* UNIXWORLD*'s "Library" column and is the Advisor Emeritus to Uni-Ops Books in Boonville, Calif.*

Books That Get You Started

Introducing the UNIX System
McGraw-Hill
800-262-4729
556 pages
Coverage*: Version 7, Bourne, vi/ex, nroff

The UNIX Operating System
second edition
Wiley
212-850-6000
455 pages
Coverage: System V/BSD, Bourne, vi, none

UNIX for People
Prentice Hall
201-767-5937
528 pages
Coverage: BSD, C, vi, nroff/troff

A Practical Guide to UNIX System V
second edition
Benjamin/Cummings
800-447-2226 or 617-944-3700
700 pages
Coverage: SVR4, all shells, vi, nroff

Peter Norton's Guide to UNIX
Bantam
212-765-6500
560 pages
Coverage: Any version, Bourne/C, vi, none

Using UNIX
Que
317-573-2500
693 pages
Coverage: SVR3.2, Bourne, ed, none

UNIX System V Release 4: An Introduction
Osborne/McGraw-Hill
510-548-2805
1,228 pages
Coverage: SVR4, Bourne, ed/vi, troff

Using SCO UNIX
Osborne/McGraw-Hill
510-548-2805
610 pages
Coverage: SCO, Bourne/C, vi, nroff

**Coverage notes the specific versions of operating system, shell, editor, and text formatter that are primarily covered by each of these books.*

Living the UNIX Life

While you're using one of the books discussed in this article to learn the technical side of UNIX, don't neglect the broader aspects of the system.

Over the decades, a tradition, community, and style have developed around UNIX. The most pleasant way to introduce yourself to all this is to pick up a copy of *Life with UNIX: A Guide for Everyone* by Don Libes and Sandy Ressler, published by Prentice Hall.

Their book provides plenty of history, mixed with tidbits of gossip, as well as guidance toward printed information, industry events, the Internet, services, and application packages. An abundance of contact information is provided along with realistic evaluations of what's mentioned. There's a chapter on how UNIX is faring in the larger world of general computing that features the cons as well as the pros of the UNIX system, and another that fearlessly predicts future directions. Technical aspects of UNIX are overviewed in three chapters.

Best of all, it's a no-pain, no-strain read. The writing style is breezy and buoyant. Even if I weren't involved with UNIX, I could see myself reading this as an evening's diversion – it's more gripping than the average paperback novel. I doubt I'll ever see this book's equal as an entre into a computer user community. *– W. Z.*

REVIEW

Emacs - The Extensible Self-documenting Text Editor

A Public Domain Software Review

By Roland Schneider

The emacs editor, in various incarnations, has been around for a long time. I'm going to talk about the freely available GNU emacs. GNU emacs can best be described as a "power user's" editor – it isn't particularly easy to learn, but it is very efficient to use after a little practice. Emacs is interesting in that the program itself is mostly an interpreter for a dialect of LISP, in which most of the actual editing functions are written. Anyone can add their own LISP code – hence the "extensible" part of the title. Emacs is self-documenting because all the documentation is available on-line from within the editor.

Because GNU emacs is so big and takes a while to start up, it is usually left running, either in a window if you're using a window system or stopped in the background with a `Ctrl-Z` (if your version of UNIX supports job control). Emacs can handle many files simultaneously, allowing you to cut and paste from one to another. You can split your screen into several windows, either for multiple views of a single file, or views of multiple files, or any combination you choose. Of course, this is most useful if you have a large number of lines on your screen to begin with. (I use a 50 line X-window)

Unlike vi, emacs has no command mode – commands are triggered by a series of one or more keystrokes, starting with a control character. You can bind any command to any sequence of keystrokes you like, including the function keys. Commands not bound to any key can be executed by typing `Meta-X`, followed by the command's name and arguments. Common commands, like "kill-line", are bound to single keys, like "`Ctrl-K`", while less-used operations, like "find-file-other-window", which opens a file in another editing window, are bound to multiple keystroke sequences, in this case "`Ctrl-X 4 f`". Mouse-based functions are supported under X-windows.

Interesting features

Emacs has a lot of useful, and unusual, features. It's standard search method is incremental, meaning that the search is performed as you type the characters you want to match. This makes it easy to find the text sequence you want with the minimum amount of typing.

Emacs understands the syntax of what you are editing. If you are working with a C program, for example, the tab key is automatically redefined to indent your code according to its nesting level, making it easy to spot incorrectly placed parentheses and braces. You can set up the indenting style to whatever you are accustomed to. Emacs also understands syntactic structures so you can move to the beginning or end of a function or a code block with a single keystroke. When you are typing closing brackets, braces, or parentheses, the cursor will momentarily jump to the matching opening symbol, making it easy to correctly match deeply nested expressions.

When you are ready to compile your program, you can do it from within emacs. Emacs interprets the messages produced by the compiler, so you can jump to successive errors by simply typing "`Ctrl-X ``", even if you compiled multiple files with "make". Emacs can do the same thing with the pattern matching program "grep".

Emacs has many many more features than I can possibly present here. I don't even use most of its capabilities. There are some people who effectively use emacs as their shell, doing everything from reading e-mail and network news to writing, compiling, and testing their programs from within the editor. For someone who does a lot of editing, and is tired of the limitations and old-fashioned command structure of vi, I would recommend having a look at GNU emacs.

Facts in Brief

Runs on:	almost any UNIX machine and VMS
How to get it:	lots of ftp sites, MUUG PD software tape (when we put it together)
Disk space:	9.9 MB for the source, 7.6 MB for executable, LISP libraries, and documentation
Usefulness:	High
Learning:	Hard (as cryptic as vi, but has more commands)
Installation effort:	Moderate
Cost:	Free

Fall Seminar Announcement

Open Systems - Getting Past the Hype

Thursday, November 19, 1992

Winnipeg Convention Centre

Stay tuned for the joint MUUG and CIPS (Canadian Information Processing Society) fall seminar called *Open Systems - Getting Past the Hype*. This event is not to be missed! Keynote speaker, Mr. Tom Wheeler, author of the *Open Systems Handbook*, will provide participants with the Open System vision of today and tomorrow. Following this presentation will be four case studies provided by company/institution representatives dealing with their move to Open Systems. Such subjects as dealing with the bureaucracy, how to begin, what are the various views and more, will be discussed. The seminar will conclude with a panel discussion with participants having the opportunity to ask questions of the various presenters. See next month's newsletter for more in-depth information.

For more information, call Susan Zuk at 788-7312.

Using Shared Memory for Inter-Process Communication

Part 3 of 3

By Peter Graham

Many moons ago now, I introduced you to the Unix Sys V shared memory facilities. I hope you haven't forgotten everything since the bulk of this article is simply code. If you have forgotten things you had best check your back issues since space prevents me from re-iterating the basics.

In this article I will provide you with some sample code for a print spooling application using shared memory as the inter-process communication medium. The example is only moderately contrived and should make reasonably good sense to most everyone.

The spooling system consists of a client program which allows users to specify files to be printed (just as you might with **lpr**) and two server programs. There is a spool server which interacts with the client(s) through a shared memory buffer to spool the files to be printed and a print server which interacts with the spool server to actually "print" the spooled files. (In the interest of brevity, and saving trees, this application only prints to stdout.) Interaction between the spool and print servers is also through a shared memory buffer. For illustrative purposes, data is stored in one buffer as ASCII characters and in the other using the appropriate binary encodings. This freedom of choice in data representation (without hassles) is another advantage of using shared memory over, say, Unix domain sockets or RPC.

The buffer between the client and spool server is referred to as the "shared record buffer" and is logically owned by the spool server. The buffer between the spool and print servers is referred to as the "shared filename buffer", for reasons that will later become clear, and it "belongs" to the print server. The owner of a given buffer assumes the responsibility for creating and initializing it.

Naturally enough, access to the shared buffers is controlled through the use of semaphores.

At any given point in time, several print clients may be running concurrently. It is their job to read the data in the files specified by the user and copy (i.e. "spool") it one line per record into the shared record buffer for later extraction by the spool server. (It is assumed that the data to be printed is printable and linefeed delimited into reasonably small pieces. A simple labelling protocol is used to distinguish between records from different files within the buffer since, due to the concurrency, they may be intermixed. This protocol also supports the passing such information as the name of the file, its size, the user who requested the print, etc. to the spool server process.

The spool server removes records from the shared record buffer and accumulates them to spool files. (There is one spool file for each file being printed.) Once all the records have been received for a given file, the spool server creates an entry in the shared filename buffer containing the name of the corresponding spool file and the print information received from the client.

The print server removes entries from the shared filename buffer and adds them to an internal list it maintains of outstanding work. It then prints the spool files on that list in an order determined by the size of the file and how long it has been waiting to print. A typical header page is prepended to each file as it is printed which contains the file name, print requestor, etc.

For better or for worse, here is the code:

```
/****************/
/* sh_rec_buf.h */
/****************/

/* the shared record buffer consists of 250
 * elements of 600 characters each and is
 * implemented as a circular queue. */

#define NUM_SH_REC_ELTS 250
#define ELTSIZE 600

/* a buffer element */
typedef char SH_REC_BUF_ELT[ELTSIZE];

typedef struct srb {
      int   head, /* head pointer in queue */
            tail, /* tail pointer in queue */
            used; /* number of elts in queue */
      /* the elements themselves */
      SH_REC_BUF_ELT   elts[NUM_SH_REC_ELTS];
} SH_REC_BUF, *SH_REC_BUF_PTR;

/****************/
/* sh_rec_buf.c */
/****************/

#include <strings.h>
#include "sh_rec_buf.h"

/*
 * srb_init: This routine initializes the shared
 * record buffer. It must be called before either
 * the spool server or any client attempts to
 * access the buffer.
 */

srb_init(qptr)
SH_REC_BUF_PTR   qptr;
{
      qptr->head=0;
      qptr->tail=0;
      qptr->used=0;
} /* srb_init */

/*
 * srb_insert: This routine inserts a new element
 * (bufptr) into the shared record buffer (qptr).
 * Synchronized access to the shared buffer is
 * assured using the associated semaphore (sema).
 */

srb_insert(sema,qptr,bufptr)
int              sema;
SH_REC_BUF_PTR   qptr;
SH_REC_BUF_ELT   *bufptr;
{
      /* get exclusive access to shared segment */
      sem_wait(sema);

      /* make sure enough space to insert record*/
      while (qptr->used==NUM_SH_REC_ELTS) {
            /* buffer full so process must */
            /* release semaphore, sleep */
            /* and try again later */
            sem_signal(sema);
            sleep(1);
            sem_wait(sema);
      }

      /* insert the record */
      strcpy(qptr->elts[qptr->head],bufptr);
      qptr->head=(qptr->head+1)%NUM_SH_REC_ELTS;
      qptr->used++;

      /* release access to the shared segment */
      sem_signal(sema);
} /* srb_insert */

/*
 * srb_remove: This routine removes an element
 * (bufptr) from the shared record buffer (qptr).
 * Synchronized access to the shared buffer is
 * assured using the associated semaphore (sema).
 */

srb_remove(sema,qptr,bufptr)
int              sema;
SH_REC_BUF_PTR   qptr;
SH_REC_BUF_ELT   *bufptr;
{
      /* get exclusive access to shared segment */
      sem_wait(sema);

      /* make sure there's a record to remove */
      while (qptr->used==0) {
            /* buffer empty so process must */
            /* release semaphore, sleep, */
            /* and then try again  */
            sem_signal(sema);
            sleep(1);
            sem_wait(sema);
      }

      /* remove and return the record */
      strcpy(bufptr,qptr->elts[qptr->tail]);
      qptr->tail=(qptr->tail+1)%NUM_SH_REC_ELTS;
      qptr->used--;

      /* release access to the shared segment */
      sem_signal(sema);
} /* srb_insert */

/******************/
/* sh_fname_buf.h */
/******************/

/* the shared filename buffer consists of 25 */
/* elements, implemented as a circular queue. */

#define NUM_SH_FNAME_ELTS 25
```

```
typedef struct fd {
      char  filename[128];
      char  spoolname[256];
      char  requestor[16];
      int   filesize;
      long  submit_time;
} SH_FNAME_BUF_ELT;

typedef struct sfb {
      int               head,
                        tail,
                        used;
      SH_FNAME_BUF_ELT elts[NUM_SH_FNAME_ELTS];
} SH_FNAME_BUF, *SH_FNAME_BUF_PTR;

/******************/
/* sh_fname_buf.c */
/******************/

#include <strings.h>
#include "sh_fname_buf.h"

/*
 * sfb_init: This routine initializes the shared
 * filename buffer. It must be called before
 * either the print server or the spool server
 * attempts to access the buffer.
 */

sfb_init(qptr)
SH_FNAME_BUF_PTR qptr;
{
      qptr->head=0;
      qptr->tail=0;
      qptr->used=0;
} /* sfb_init */

/*
 * sfb_insert: This routine inserts a new element
 * (fdptr) into the shared filename buffer (qptr).
 * Synchronized access to the shared buffer is
 * assured using the associated semaphore (sema).
 */

sfb_insert(sema,qptr,fdptr)
int             sema;
SH_FNAME_BUF_PTR qptr;
SH_FNAME_BUF_ELT *fdptr;
{
      /* get exclusive access to shared segment */
      sem_wait(sema);

      /* make sure enough space to insert record*/
      while (qptr->used==NUM_SH_FNAME_ELTS) {
            /* buffer full so process must */
            /* release semaphore, sleep */
            /* and try again later */
            sem_signal(sema);
            sleep(1);
            sem_wait(sema);
      }

      /* insert the record */
      qptr->elts[qptr->head]=*fdptr;
      qptr->head=(qptr->head+1)%NUM_SH_FNAME_ELTS;
      qptr->used++;

      /* release access to the shared segment */
      sem_signal(sema);
} /* sfb_insert */

/*
 * sfb_remove: This routine removes an element
 * (fdptr) from the shared filename buffer (qptr).
 * Synchronized access to the shared buffer is
 * assured using the associated semaphore (sema).
 */

int sfb_remove(sema,qptr,fdptr,num_to_print)
int             sema;
SH_FNAME_BUF_PTR qptr;
SH_FNAME_BUF_ELT *fdptr;
int             num_to_print;
{
      /* get exclusive access to shared segment */
      sem_wait(sema);

      /* make sure there's a record to remove */
      if ((qptr->used==0) && (num_to_print!=0)) {
            /* return to caller so we don't     */
            /* block the printing of outstanding*/
            /* jobs waiting for another request.*/
            return(0);
      }
      while (qptr->used==0) {
            /* buffer empty so process must */
            /* release semaphore, sleep, */
            /* and then try again  */
            sem_signal(sema);
            sleep(1);
            sem_wait(sema);
      }

      /* remove and return the record */
      *fdptr=qptr->elts[qptr->tail];
      qptr->tail=(qptr->tail+1)%NUM_SH_FNAME_ELTS;
      qptr->used--;

      /* release access to the shared segment */
      sem_signal(sema);
      return(1);
} /* sfb_insert */

/************/
/* client.c */
/************/

#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <errno.h>
#include <pwd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "sh_rec_buf.h"

/* necessary because of problem */
/*with SunOS include file shm.h */
#define SHM_W 0200
#define SHM_R 0400

/* inserts records into shared record buffer */
extern srb_insert();

main(argc,argv)
int   argc;
char  *argv[];
{
      struct passwd    *pw_ptr;
                  /* ptr to passwd structure */
      unsigned    uid;  /* user's numeric userid*/
      FILE        *in_f;/* input file to read */
      char        read_buf[513];
                  /* input file buffer */
      int         total_bytes;
                  /* input file bytes count */
      int         argcnt;
                  /* current argument to process*/
      SH_REC_BUF_ELT    data_record;
                  /* record to insert into */
                  /* shared record buffer */
      long        request_time;
                  /* time of request: together  */
                  /* with filename this uniquely*/
                  /* identifies a print request.*/
      key_t       srb_shmkey,
                  /* key to shared mem. segment */
                  srb_semakey;
                  /* key to semaphore */
      int         srb_segid,
                  /* shared memory segment ID */
                  srb_sema;
                  /* the access semaphore */
      SH_REC_BUF_PTR    srb_segaddr;
                  /* pointer to shared buffer */
                  /* stored in shared mem. seg. */

      /* verify that some files were specified */
      if (argc<=1) {
            printf(
"client: No args specified. Nothing printed.\n");
            exit(-1);
      }

      /* get user info and current time */
      uid=getuid();
      pw_ptr=getpwuid(uid);

      /* create semaphore to control mem. access*/
      if ((srb_semakey=ftok(
                  "/muug/shm3/spool_server.c",
                  'S'))==-1) {
            printf(
      "client: Couldn't create semaphore key.\n");
            exit(-1);
      }
      if ((srb_sema=sem_create(srb_semakey,
                                    1))==-1) {
            printf(
      "client: Couldn't create the semaphore.\n");
            exit(-1);
      }

      /* setup to access shared mem. record buf.*/
      if ((srb_shmkey=ftok(
                  "/muug/shm3/spool_server.c",
                  'M'))==-1) {
            printf(
  "client: Couldn't create shared memory key.\n");
            exit(-1);
      }
      if ((srb_segid=shmget(srb_shmkey,
                        sizeof(SH_REC_BUF),
                        SHM_R|SHM_W))==-1) {
            printf(
"client: Couldn't get shared memory segment.\n");
            exit(-1);
      }
      if ((srb_segaddr=(SH_REC_BUF_PTR)
            shmat(srb_segid,(char *)0,0))==
          (SH_REC_BUF_PTR) (-1)) {
            printf(
"client: Couldn't attach shared memory seg.\n");
            exit(-1);
      }

      /* process each argument (file to print) */
      for (argcnt=1;argcnt<argc;argcnt++) {
            /* check this argument */
            if (-1==access(argv[argcnt],R_OK)) {
                  /* Bad file specified */
                  if (errno==ENOENT) {
                        printf(
                    "client: The file %s does not exist.\n",
                              argv[argcnt]);
                        exit(-1);
                  } else if (errno==EACCES) {
                        printf(
                   "client: The file %s is inaccessible.\n",
                              argv[argcnt]);
                        exit(-1);
                  } else {
                        printf(
                     "client: Argument number %d is bad.\n",
                              argcnt);
                        exit(-1);
                  }
            }
            /* get the current (request) time */
            request_time=time(0);

            /* Create a header record to insert */
            /* into the shared buffer */
            sprintf(data_record,
                 "HDR %s %s %ld\n\0",argv[argcnt],
                  pw_ptr->pw_name,request_time);
            srb_insert(srb_sema,srb_segaddr,
                  data_record);

            /* Read the "records" in specified */
            /* file to be printed */
            in_f=fopen(argv[argcnt],"r");
            total_bytes=0;
            while ((fgets(read_buf,
                        sizeof(read_buf),in_f))
                              !=NULL) {
                  /* keep sum of record lengths */
                  total_bytes+=strlen(read_buf);
                  /* Form data record to insert */
                  /* into the shared buffer */
                  sprintf(data_record,
                        "DAT %s %d %ld :\0",
                        argv[argcnt],
                        strlen(read_buf),
                        request_time);
                  strcat(data_record,read_buf);
                  srb_insert(srb_sema,srb_segaddr,
                        data_record);
            }

            /* Form end record to insert into */
            /* the shared buffer */
            sprintf(data_record,"END %s %d %ld\0",
                  argv[argcnt],total_bytes,
                  request_time);
            srb_insert(srb_sema,srb_segaddr,
                  data_record);
            fclose(in_f);
      }

      /* shut down our connection to shared buf.*/
      if (shmdt((char *)srb_segaddr)==-1) {
            printf(
"client: Couldn't detach shared memory seg.\n");
            exit(-1);
      }

      /* close the semaphore */
      sem_close(srb_sema);
} /* main */

/******************/
/* spool_server.h */
/******************/

/* this structure describes a file requested */
/* for spooling. It is used by the spool server */
/* to keep track of all the files it is spooling*/
/* for potentially many clients. */
/* A hashed bucketting scheme is used to allow */
/* us to find particular print requests quickly.*/
/* Within each bucket, a doubly linked list */
/* is used for easy deletion of elements. */

/* a convenient small prime for hash function */
#define HASHTABSIZE 97

typedef struct sr {
      char  filename[128];
            /* name of file to be printed */
      char  spoolname[256];
            /* the spoolfile's name */
      int   spoolfd;
            /* file descriptor for spool file */
```

```
      long  submit_time;
            /* time when print request made */
      char  requestor[16];
            /* userid of requestor */
      struct sr   *next,
                  *prev;
}SPOOLREC, *SPOOLRECPTR;

typedef struct ht {
      SPOOLRECPTR buckets[HASHTABSIZE];
} HASHTABLE;

/******************/
/* spool_server.c */
/******************/

#include <stdio.h>
#include <strings.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "sh_rec_buf.h"
#include "sh_fname_buf.h"
#include "spool_server.h"

/* necessary because of problem  */
/* with SunOS include file shm.h */
#define SHM_W 0200
#define SHM_R 0400

#define SPOOLDIR "/tmp/"

extern int srb_remove();
extern int sfb_insert();

static HASHTABLE spooltab;
                  /* hash table for spool info */
static int        print_server_pid;

static key_t      srb_shmkey,
                  srb_semakey;
static int        srb_segid,
                  srb_sema;
static SH_REC_BUF_PTR        srb_segaddr;
static struct shmid_ds       *srb_shmbufptr;
static key_t      sfb_shmkey,
                  sfb_semakey;
static int        sfb_segid,
                  sfb_sema;
static SH_FNAME_BUF_PTRsfb_segaddr;

/*
 * handler - signal handler to permit graceful
 * killing of the server processes.
 */

handler()
{
      /* print out msg so we know we're stopping*/
      printf(
"***Spooler shutting down on SIGQUIT signal.\n");
      fflush(stdout); /* make sure it's flushed */

      /* signal print server to quit as well */
      kill(print_server_pid,SIGQUIT);

      /* clean up after ourselves */
      if (shmdt((char *) srb_segaddr)==-1) {
            printf(
"spool_server: Couldn't detach record buffer.\n");
      }

      if (shmdt((char *) sfb_segaddr)==-1) {
            printf(
"spool_server: Couldn't detach filename buf.\n");
      }
      if (shmctl(srb_segid,IPC_RMID,srb_shmbufptr)
                                        ==-1) {
            printf(
"spool_server: Couldn't remove record buffer.\n");
      }
      /* remove semaphore since it's */
      /* under our control */
      sem_rm(srb_sema);
      sem_close(sfb_sema);

      /* exit normally */
      exit(0);
} /* handler */

/*
 * hash - implements simple-minded hash function.
 */

int hash(sptr)
char  *sptr;
{
      int   hashval;

      hashval=0;
      for (sptr;(*sptr)!='\0';sptr++) {
            /* sum in each character value */
            hashval=hashval+*sptr;
      }
      /* return value mod HASHTABSIZE */
      return (hashval%HASHTABSIZE);
} /* hash */

/*
 * process_header - processes a header record
 * received in the shared record buffer. This
 * involves creating a spool record for it,
 * loading the record from the info. in the
 * header record, putting the record into the
 * appropriate hash bucket, and creating
 * a spool file to write the data records into.
*/

int process_header(buf)
char  buf[512];
{
      char        filename[128],
                  username[16];
      long        submit_time;
      SPOOLRECPTR srecptr,
                  curr,
                  prev;
      int         hashval;
      char        tmpstr[16];

      /* extract file info. from buf */
      sscanf(buf,"HDR %s %s %ld",
            filename,username,&submit_time);

      /* create a record holding information */
      /* from the HDR record */
      srecptr=(SPOOLRECPTR)
                  malloc(sizeof(SPOOLREC));
      strcpy(srecptr->filename,filename);
      strcpy(srecptr->requestor,username);
      srecptr->submit_time=submit_time;
      srecptr->next=NULL;
      srecptr->prev=NULL;

      /* form spool file name using the */
      /* original filename, the userid of the */
      /* print requestor, and the current time. */
      strcpy(srecptr->spoolname,SPOOLDIR);
      strcat(srecptr->spoolname,username);
      strcat(srecptr->spoolname,"_");
      strcat(srecptr->spoolname,filename);
      strcat(srecptr->spoolname,"_");
      sprintf(tmpstr,"%ld",submit_time);
      strcat(srecptr->spoolname,tmpstr);

      /* create the spool file and leave */
      /* the open file descriptor in the */
      /* corresponding spool structure */
      if ((srecptr->spoolfd=creat(
                              srecptr->spoolname,
                              0600)) == -1) {
            printf(
"spool_server: Couldn't create spool file.\n");
            exit(-1);
      }

      /* hash to the appropriate bucket */
      hashval=hash(filename);

      /* insert at end of that list */
      prev=NULL;
      for (curr=spooltab.buckets[hashval];
                  curr!=NULL;curr=curr->next) {
            prev=curr;
      }
      if (prev==NULL) {
            /* first element on list */
            spooltab.buckets[hashval]=srecptr;
      } else {
            prev->next=srecptr;
            srecptr->prev=prev;
      }
} /* process_header */

/*
 * process_data - processes a data record from
 * the shared record buffer by extracting it,
 * finding the spool record for the corresponding
 * file to determine what spool file to write the
 * data to and then writing the data to that file.
 */

int process_data(buf)
char  buf[512];
{
      char        filename[128];
      int         recsize;
      long        submit_time;
      char        text[512];
      SPOOLRECPTR curr;
      int         hashval;
      char        *sptr;
      int         tidx;

      /* extract file info. from buf */
      sscanf(buf,"DAT %s %d %ld",
            filename,&recsize,&submit_time);
      tidx=0;
      sptr=index(buf,':');
      for (sptr++;*sptr!='\0';sptr++) {
            text[tidx++]=*sptr;
      }
      text[tidx]='\0';

      /* find the correct bucket */
      hashval=hash(filename);

      /* Scan down the list to find the spool */
      /* record in question. Both the        */
      /* filename and submit time must match. */
      for (curr=spooltab.buckets[hashval];
            strcmp(filename,curr->filename)
               &&(submit_time==curr->submit_time);
            curr=curr->next) {
      }

      /* copy this record to appropriate file */
      write(curr->spoolfd,text,recsize);
} /* process_data */

/*
 * process_end - processes an END record from the
 * shared record buffer.  This involves closing
 * the corresponding spool file and creating and
 * inserting a record for the print server into
 * the shared filename buffer.
 */

int process_end(buf)
char  buf[512];
{
      char        filename[128];
      int         filesize;
      long        submit_time;
      SPOOLRECPTR curr;
      int         hashval;
      SH_FNAME_BUF_ELT rec;

      /* extract file info. from buf */
      sscanf(buf,"END %s %d %ld",
            filename,&filesize,&submit_time);

      /* find the correct bucket */
      hashval=hash(filename);

      /* Scan down the list to find the spool */
      /* record in question. Both the        */
      /* filename and submit time must match. */
      for (curr=spooltab.buckets[hashval];
            strcmp(filename,curr->filename)
               &&(submit_time==curr->submit_time);
            curr=curr->next) {
      }

      /* close so it can be opened by server */
      close(curr->spoolfd);

      /* format and write a print request to */
      /* shared filename buffer */
      strcpy(rec.filename,curr->filename);
      strcpy(rec.spoolname,curr->spoolname);
      strcpy(rec.requestor,curr->requestor);
      rec.filesize=filesize;
      rec.submit_time=curr->submit_time;
      sfb_insert(sfb_sema,sfb_segaddr,&rec);

      /* delete the spool record from the list */
      if (curr->next!=NULL) {
            curr->next->prev=curr->prev;
      }
      if(curr->prev!=NULL) {
            curr->prev->next=curr->next;
      } else {
            spooltab.buckets[hashval]=curr->next;
      }
} /* process_end */

main(argc,argv)
int   argc;
char  *argv[];
{
      int               i;
      SH_REC_BUF_ELT    data_record;

      /* extract pid of print server from */
      /* argv[1] and store it */
      sscanf(argv[1],"%d",&print_server_pid);

      /* setup signal handler to catch SIGQUIT */
      signal(SIGQUIT,handler);

      /* Initialize hash buckets - no entries */
      for (i=0;i<HASHTABSIZE;i++) {
            spooltab.buckets[i]=NULL;
      }

      /* create a semaphore to control access */
      /* to shared record buffer */
      if ((srb_semakey=ftok(
                  "/muug/shm3/spool_server.c",
                  'S'))==-1) {
            printf(
"spool_server: Couldn't create srb sem. key.\n");
            exit(-1);
```

```
      }
      if ((srb_sema=sem_create(srb_semakey,1))
                                        ==-1) {
            printf(
"spool_server: Couldn't create srb semaphore.\n");
            exit(-1);
      }

      /* setup to access shared mem. record buf.*/
      if ((srb_shmkey=ftok(
                  "/muug/shm3/spool_server.c",
                  'M'))==-1) {
            printf(
"spool_server: Couldn't create srb seg. key.\n");
            exit(-1);
      }
      if ((srb_segid=shmget(srb_shmkey,
                        sizeof(SH_REC_BUF),
                        IPC_CREAT|SHM_R|SHM_W))
                  ==-1) {
            printf(
"spool_server: Couldn't create the srb seg.\n");
            exit(-1);
      }
      if ((srb_segaddr=(SH_REC_BUF_PTR)
                  shmat(srb_segid,(char *)0,0))
               ==(SH_REC_BUF_PTR) (-1)) {
            printf(
"spool_server: Couldn't attach shrd.mem. seg.\n");
            exit(-1);
      }

      /* create a semaphore to control access */
      /* to the shared filename buffer */
      if ((sfb_semakey=ftok(
                  "/muug/shm3/print_server.c",
                  'S'))==-1) {
            printf(
"spool_server: Couldn't create sfb sem. key.\n");
            exit(-1);
      }
      if ((sfb_sema=sem_create(sfb_semakey,1))
                  ==-1) {
            printf(
"spool_server: Couldn't create the sfb sem.\n");
            exit(-1);
      }

      /* setup to access shared filename buf. */
      if ((sfb_shmkey=ftok(
                  "/muug/shm3/print_server.c",
                  'M'))==-1) {
            printf(
"spool_server: Couldn't create sfb seg. key.\n");
            exit(-1);
      }
      if ((sfb_segid=shmget(sfb_shmkey,
                        sizeof(SH_FNAME_BUF),
                        SHM_R|SHM_W))==-1) {
            printf(
"spool_server: Couldn't create the sfb seg.\n");
            exit(-1);
      }
      if ((sfb_segaddr=(SH_FNAME_BUF_PTR)
                  shmat(sfb_segid,(char *)0,0))
               ==(SH_FNAME_BUF_PTR) (-1)) {
            printf(
"spool_server: Couldn't attach sfb segment.\n");
            exit(-1);
      }

      /* initialize the shared record buffer */
      srb_init(srb_segaddr);

      /* repeatedly read from shared record buf.*/
      /* servers almost always run infinite loop*/
      for (;;) {
            srb_remove(srb_sema,srb_segaddr,
                  data_record);
            switch (data_record[0]) {
            case 'H': /* a Header record */
                  process_header(data_record);
                  break;
            case 'D': /* a Data record */
                  process_data(data_record);
                  break;
            case 'E': /* an End record */
                  process_end(data_record);
                  break;
            }
      }
} /* main */

/******************/
/* print_server.h */
/******************/

/* Structure describes print job to be done. */
/* A doubly linked list of them is maintained */
/* to permit easy deletion. */
typedef struct pr {
      char  filename[128];    /* orig. filename */
      char  spoolname[256];   /* spool filename */
      long  submit_time;      /* time job was */
                              /* "submitted" */
      int   filesize;         /* print file size*/
      char  requestor[16];    /* his/her ID */
      int   prio;             /* print priority */
      struct pr   *next,
                  *prev;
} PRINTREC, *PRINTRECPTR;

/******************/
/* print_server.c */
/******************/

#include <stdio.h>
#include <strings.h>
#include <time.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "sh_fname_buf.h"
#include "print_server.h"

/* necessary because of problem  */
/* with SunOS include file shm.h */
#define SHM_W 0200
#define SHM_R 0400

#define SPOOLDIR "/tmp/"

static PRINTRECPTR     printqptr;
static key_t      sfb_semakey,
                  sfb_shmkey;
static int        sfb_segid,
                  sfb_sema;
static SH_FNAME_BUF_PTRsfb_segaddr;
static struct shmid_ds *sfb_shmbufptr;

/*
 * handler - signal handler to permit graceful
 * killing of the server processes.
 */

handler()
{
      /* print message so we know we're stopping*/
      printf(
"***Server shutting down on spooler's signal.\n");
      fflush(stdout); /* make sure it's flushed */

      /* sleep for a bit to allow spool server */
      /* to let go of the shared filename */
      /* buffer and semaphore */
      sleep(1);

      if (shmdt((char *) sfb_segaddr)==-1) {
            printf(
"print_server: Couldn't detach filename buf.\n");
      }
      if (shmctl(sfb_segid,IPC_RMID,
                        sfb_shmbufptr)==-1) {
            printf(
"print_server: Couldn't remove record buf.\n");
      }
      /* remove semaphore since it's */
      /* under our control */
      sem_rm(sfb_sema);

      /* exit normally */
      exit(0);
} /* handler */

/*
 * print_prio - calculate print priority of a job.
 */

int print_prio(size,time_waiting)
int   size;
long  time_waiting;
{
      /* Print priority should increase with */
      /* waiting time and decrease with the  */
      /* size of the print job! */

      /* The value 10000 approximates 10 pages  */
      /* of output. Jobs of more than 10000     */
      /* characters will automatically have the */
      /* lowest priority and will be printed    */
      /* FCFS after all smaller jobs. Smaller   */
      /* jobs will be assigned priorities based */
      /* on their relative sizes. */
      return ((int)((float)(100000.0/size)
                              *time_waiting)%100);
}

/*
 * process_new_request - This routine processes a
 * new print request once it has been removed from
 * the shared filename buffer. It does this by
 * queueing up the request info. on a print queue.
 */

int process_new_request(frecptr)
SH_FNAME_BUF_ELT *frecptr;
{
      char        filename[128];
      char        spoolname[256];
      char        username[16];
      int         filesize;
      long        submit_time;
      PRINTRECPTR precptr,
                  curr,
                  prev;

      /* add print job to print queue */
      precptr=(PRINTRECPTR)
                  malloc(sizeof(PRINTREC));
      strcpy(precptr->filename,frecptr->filename);
      strcpy(precptr->spoolname,
                  frecptr->spoolname);
      strcpy(precptr->requestor,
                  frecptr->requestor);
      precptr->filesize=frecptr->filesize;
      precptr->submit_time=frecptr->submit_time;
      precptr->next=NULL;
      precptr->prev=NULL;
      if (printqptr==NULL) {
            /* first entry on list */
            printqptr=precptr;
      } else {
            /* add onto existing list at end so */
            /* big print jobs will get FCFS */
            /* service. (i.e. no starvation) */
            prev=NULL;
            for (curr=printqptr; curr!=NULL;
                        curr=curr->next) {
                  prev=curr;
            }
            precptr->prev=prev;
            prev->next=precptr;
      }
} /* process_new_request */

/*
 * select_and_print - This routine selects the
 * next print job from the print queue using the
 * print_prio function to make its choice. Once
 * selected the job is "printed" (to the screen)
 * complete with a header page.
 */

int select_and_print()
{
      PRINTRECPTR prtptr,
                  currprtjobptr;
      int         num_on_queue,
                  highprio;
      FILE        *in_f;
      char        ch;
      long        currtime,
                  seconds,
                  minutes,
                  hours,
                  time_waiting;

      /* select entry to print - must scan all */
      /* jobs on print queue and calculate */
      /* function current waiting time, then */
      /* apply priorization using waiting time */
      /* and file size to select a job */
      currtime=time(0);

      num_on_queue=0;
      /* start by calculating current print */
      /* priorities and counting the number of */
      /* outstanding print jobs on the queue */
      for (prtptr=printqptr;prtptr!=NULL;
                        prtptr=prtptr->next) {
            prtptr->prio=
                  print_prio(prtptr->filesize,
                  currtime - prtptr->submit_time);
            num_on_queue++;
      }

      /* now rescan print queue to */
      /* find highest priority job */
      highprio=-1;
      for (prtptr=printqptr;prtptr!=NULL;
                        prtptr=prtptr->next) {
            if (prtptr->prio>highprio) {
                  highprio=prtptr->prio;
                  currprtjobptr=prtptr;
            }
      }

      /* remove this print job from queue */
      if (currprtjobptr->next!=NULL) {
            currprtjobptr->next->prev=
                  currprtjobptr->prev;
      }
      if (currprtjobptr->prev!=NULL) {
            currprtjobptr->prev->next=
                  currprtjobptr->next;
      } else {
            printqptr=currprtjobptr->next;
      }

      /* print banner page */
      printf(
"\f\n\n\n\n\n\n\n\n\n----------------------------------
-----------------------\n");
      printf("|  Filename: %s\n",
            currprtjobptr->filename);
```

```
      printf("|  Requestor: %s\n",
            currprtjobptr->requestor);
      printf("|  Size in bytes: %d\n",
            currprtjobptr->filesize);
      printf("|  Print priority: %d\n",
            currprtjobptr->prio);
      printf("|  Print Time: %s",
            ctime(&currtime));
      printf("|  Time spent waiting to print: ");
      time_waiting=
            currtime-currprtjobptr->submit_time;
      seconds=time_waiting%(long) 60;
      time_waiting-=seconds;
      minutes=(time_waiting%(long) 3600);
      time_waiting-=minutes;
      minutes=minutes/(long) 60;
      hours=time_waiting/(long) 3600;
      if (hours!=0) {
            if (hours==1) {
                  printf("%ld hour ",hours);
            } else {
                  printf("%ld hours ",hours);
            }
      }
      if (minutes!=0) {
            if (minutes==1) {
                  printf("%ld minute ",minutes);
            } else {
                  printf("%ld minutes ",minutes);
            }
      }
      if (seconds!=0) {
            if (seconds==1) {
                  printf("%ld second ",seconds);
            } else {
                  printf("%ld seconds ",seconds);
            }
      }
      printf(
"\n-------------------------------------------------
-------\n\f");

      /* write spool file to output */
      /* can assume it exists and is readable. */
      in_f=fopen(currprtjobptr->spoolname,"r");
      while ((ch=fgetc(in_f))!=EOF) {
            putchar(ch);
      }
      fclose(in_f);

      /* delete spool file */
      unlink(currprtjobptr->spoolname);

      /* free space allocated by queue element */
      free(currprtjobptr);

      /* return an indication of number */
      /* of print jobs still queued */
      return(--num_on_queue);
} /* select_and_print */

main()
{
      SH_FNAME_BUF_ELT filerec;
      int   num_to_print;

      /* setup signal handler to catch SIGQUIT */
      signal(SIGQUIT,handler);

      /* initially no jobs to print */
      printqptr=NULL;

      /* create semaphore to control */
      /* access to filename buffer */
      if ((sfb_semakey=ftok(
                  "/muug/shm3/print_server.c",
                  'S'))==-1) {
            printf(
"print_server: Couldn't create sfb sem. key.\n");
            exit(-1);
      }
      if ((sfb_sema=sem_create(sfb_semakey,1))
                                          ==-1) {
            printf(
"print_server: Couldn't create sfb semaphore.\n");
            exit(-1);
      }

      /* setup to access shared filename buffer */
      if ((sfb_shmkey=ftok(
                  "/muug/shm3/print_server.c",
                  'M'))==-1) {
            printf(
 "print_server: Couldn't create sfb key.\n");
            exit(-1);
      }
      if ((sfb_segid=shmget(sfb_shmkey,
                  sizeof(SH_FNAME_BUF),
                  IPC_CREAT|SHM_R|SHM_W))==-1) {
            printf(
"print_server: Couldn't create sfb segment.\n");
            exit(-1);
      }
      if ((sfb_segaddr=(SH_FNAME_BUF_PTR)
                  shmat(sfb_segid,(char *)0,0))
                ==(SH_FNAME_BUF_PTR) (-1)) {
            printf(
"print_server: Couldn't attach sfb segment.\n");
            exit(-1);
      }

      /* initialize the shared filename buffer */
      sfb_init(sfb_segaddr);

      /* initially there are no jobs to print */
      num_to_print=0;

      /* repeatedly read from shared record buf.*/
      /* servers almost always run infinite loop*/
      for (;;) {
            /* Remove an element from shared */
            /* filename buffer if there's one */
            /* there. If there isn't and there */
            /* are outstanding print jobs */
            /* ('num_to_print!=0') then */
            /* sfb_remove() will return 0. */
            /* Otherwise, it'll wait for new */
            /* request to arrive then return 1. */

            if (sfb_remove(sfb_sema,sfb_segaddr,
                  &filerec,num_to_print)) {
                  /* queue up returned request */
                  process_new_request(&filerec);
            }

            /* pick outstanding job and process */
            /* 'select_and_print' returns the */
            /* number of remaining jobs */
            num_to_print=select_and_print();
      }
} /* main */
```

Seminar Announcement

Clifford Stoll: Stalking the Wily Hacker

A Breakfast Seminar
Thursday, 7:30 AM, September 10, 1992
Winnipeg Convention Centre
Brought to you by CIPS (Canadian Information Processing Society)

The Winnipeg Chapter of the Canadian Information Processing Society (CIPS) invites you to its 1992/93 season "kick-off" seminar. A computer security seminar with a difference – hold onto your seat for an international race to catch an elusive hacker that 'll leave you breathless.

Someone breaks into your computer. What do you do? Slam the door? Call the police? Ignore the problem?

For a year, a German broke into over forty military computers around the world. By silently tracking him back, he learned that he was a spy, passing information to the Soviet KGB. Recently he was convicted of espionage.

What technique did he use to crack into computers? Where are the holes in our systems? How do you trace someone across the worldwide computer networks? Who was willing to help — who wasn't?

Cliff's talk, *Stalking the Wily Hacker*, will address these questions. A fun time is guarenteed for all.

The Cuckoo's Egg, the book describing this incident, tells the true story of tracking a spy through the maze of computer espionage.

Clifford Stoll is an astronomer by training and a computer security expert by accident. Since catching the Hanover Hacker he has become a leading authority on computer security, delivering more lectures on the subject than he cares to admit. He has given talks in the United States for both the Central Intelligence Agency (CIA) and National Security Agency (NSA) and has appeared before the U.S. Senate.

Call Susan Zuk at 788-7312 to receive a registration form, or send a cheque with the required amount payable to "**Canadian Information Processing Society**" and send to:

Canadian Information Processing Society
P.O. Box 2610, Winnipeg, Manitoba R3C 4B3

Please include the following information: Name, Registration Type (CIPS, Non-Member, Full Time Student), Fee, Company Name, Address, City, Postal Code, Phone, Fax.

Seminar fees are as follows:

CIPS Members	$35
Non-members	$45
Full Time Student	$20
Late Registration add	$ 5

(Late registration is required if postmarked after September 4, 1992)

TUUG Meeting Minutes

for
Tuesday, May 12, 1992, 7:30 PM
234B Engineering Bldg.
University of Manitoba
Ft. Garry Campus

Chair: Susan Zuk
Attendance: 47

Business meeting:
a) President's Report
- There was significant interest in MUUG's booth at the MWCS computer fest.
- The MUUG Online project is making progress.
- MUUG hopes for future joint projects with CIPS.
- The process of affiliation with UniForum is proceeding.

b) Membership Report
- TUUG currently has 84 members

c) Treasurer's Report
- MUUG has $1800 in chequing account and $8000 invested.

d) New Business
- Moved by Gilbert Detillieux, seconded by Peter Graham, that the MUUG executive be authorized to spend up to $2600 to acquire a large SCSI disk for the MUUG Online system.
  - other, preferably free, alternatives will be explored before a disk is purchased.
  - passed unanimously.

Presented topic:
Future trends at Intel – Jon Coxworth, Intel Corp.

Agenda

for
Tuesday, September 8, 1992, 7:30 PM
234B Engineering Bldg.
University of Manitoba
Ft. Garry Campus

1. Round Table 7:30

2. Business Meeting 8:00
   a) President's Report
   b) MUUG By-laws
   c) Membership Dues
   d) MUUG Online User Fees
   e) Nominations for the Board
   f) New Business

3. Break 8:20

5. Presented Topic 8:30
   An Introduction to MUUG Online
   By Roland Schneider
   With the UofM Computer Centre's donation of the use of a well-networked Sun 386i to our group, our long-awaited jump into modern computer communications is finally at hand. Many MUUG members have already taken advantage of this service. This talk will introduce the service, for those who haven't used it yet, and show you how to use it. It will also demonstrate the help facilities, e-mail, Usenet news, and networking in general, and explain how the different networks work.

6. Adjourn 9:30

Note: Please try to arrive at the meeting between 7:15 and 7:30 pm. Thank You.

Coming Up

Meeting:
Our next meeting is scheduled for Tuesday, October 13, at 7:30 PM. Meeting topic and location will be given in October's newsletter. The important items on the agenda will be the vote on the adoption of the MUUG by-laws, and the election of the board members for the 1992-1993 year. The presented topic might be part 2 of the presentation on MUUG Online, or it might be something else, depending on who and what we can line up for then. Stay tuned for an update.

Newsletter:
We might have a continuation to RPC Programming by Scott Balneaves. I also have a few "filler" articles by Roland Schneider. Apart from that, there are no other articles in the pipeline. I could use some more material, especially shorter articles – half a page to one page in length would be fine. I would like to see some more book reviews – have you read any good books lately? Monsieur Ex has also let me know that his mail-box is empty lately – please submit your questions via e-mail to <m-ex@muug.mb.ca> or by FAX to the editor.