Linux is an operating system on the move. Here are six applications that demonstrate a few of the ways it is being used, as well as some useful insights into the significance and wider context of Linux's recent and rapid growth.

Linux in Practice: An Overview of Applications

Terry Bollinger, The Mitre Corporation

Breaking out of its original niches of low-cost Unix development and Internet server support, Linux is popping up in all sorts of interesting places. Linux applications tend to be international, a direct consequence of Linus Torvalds' original use of the Internet to co-opt developers. They also tend to rely on the remarkable stability and performance of Linux, as well as on the ease with which developers of leading-edge applications and hardware can modify Linux source code.

This article describes six examples of Linux applications. Given that Linux is free, what is surprising is the degree to which cost was *not* the primary driver for the individuals and groups who selected Linux in these applications. Indeed, Linux was more often selected because its combination of reliability, performance, good tools, portability, and configurability made it a powerful tool for creating new applications quickly and effectively.

0740-7459/99/$10.00 © 1999

Publishing and Desktop Environments

The first Linux application example is from Phil Hughes, creator and publisher of *Linux Journal*. Although Linux is not generally viewed as a good platform for desktop publishing, Hughes provides insights not only into how Linux was useful but also about some of the shortfalls that he encountered.

One reason why Linux is comparatively weak in desktop environments and publishing is that until fairly recently, commercial tool vendors were not generally interested in porting their products to a "free" operating system. This situation changed significantly in 1998, as corporate software vendors began to recognize the potential of the Linux market for their products. Even so, Linux still lags significantly behind systems such as Microsoft Windows when it comes to powerful, highly integrated document handling.

According to a short hands-on study done by Ian Peters and commissioned by Eileen Boettcher, both of The MITRE Corporation, two large commercial office suites are currently available for Linux: Star Office (http://www.stardivision.com) and Applixware (http://www.applix.com). Star Office is free for personal use, while the Applixware package must be purchased. WordPerfect 8 for Linux Personal Edition is available as a free download from Corel and provides about the same capabilities as WordPerfect for Microsoft Windows.

The study also indicates that some interesting desktop releases are likely in 1999. At least two major open-source desktop products should be making their debuts: KOffice, which is part of the larger environment called KDE (K Desktop Environment; see http://www.kde.org), and GNOME (GNU Network Object Model Environment; see http://www.gnome.org). The KOffice suite is closest to completion, and will include a word processor, a spreadsheet, a presentation editor for making overheads, a chart and graph drawer, an image viewer, a vector drawing program, and an image viewer. GNOME is more technically ambitious than KDE, but when completed is expected to provide much more flexibility than more traditional desktop environments. GNOME is also notable for making extensive use of Corba 2.2 middleware standards and interfaces in its construction.

Finally, the study also mentions WINE (Wine Is Not an Emulator; see http://www.winehq.org), which is intended to allow Win32 applications to run on top of Linux. WINE is described as interesting, but not yet sufficiently far along to make it practical for executing an application such as Microsoft Office on top of Linux.

Rapid Initial Deployment Applications

Developing new software applications is a risky business. People who have worked in software for a while usually have a repertoire of stories about what happened to people, departments, or entire companies whose efforts to deploy new software took so long that they ended up missing their window of opportunity. In some extreme examples of software deployment gridlock, entire facilities such as the Denver International Airport have been delayed for years while the initial set of software for those facilities was completed or fixed.

Linux brings to bear some unique tools that can drastically reduce the time needed to get complex systems working.

Two of the application examples, those of Jorge Ocón and Jon Ashley, provide examples of the unique stability and configurability properties of Linux. They used Linux to help avoid certain aspects of software deployment gridlock while building systems that involve remote sites and potentially complex software requirements. The Ocón essay describes a high-volume, soft real-time train station application, while the Ashley essay describes a use of Linux in truly remote locations, including Antarctica and the bottom of the ocean floor.

Linux brings to bear some unique tools that can drastically reduce the time needed to get such complex systems working. In particular, its combination of an unusually stable and reliable kernel with a suite of Internet-compatible development tools makes the idea of rapid initial deployment feasible. RID is most appropriate when the needs for a system can be split into a small set of functions that absolutely must be provided for the system to be usable, plus a (generally much larger) set of "that would be nice" features that can wait for a while. Often this splitting of requirements translates into making the remote sites initially

Using Linux in Publishing

Phil Hughes, SSC, Inc.

When we started SSC, a publishing company, in 1983, we sought to offer documentation, training, and consulting to those in the Unix industry. Having worked with Unix since 1980, I was convinced that one of its greatest shortcomings was a lack of good documentation.

I was first introduced to Linux in 1993, and was impressed from the beginning. Having used Unix on everything from an 8088-based PC to an Amdahl, I expected Linux to be a cheap imitation. However, even in its 1993 incarnation, I found Linux to be solid and complete.

Looking for an Alternative

In spring 1994, SSC began having serious capacity problems with our single Unix System V, Release 2. Unix SVR2 included networking code, so our first thought was to buy the networking code for SCO Xenix and hook these two systems together. The licensing costs for each copy of SCO Xenix were steep but acceptable. However, at that time SCO Xenix was on its way out, with SCO Unix taking its place. We also heard rumors of networking problems, and we weren't sure if everything would work.

I decided to try the cost-free Linux approach as an alternative. We had little to lose, and if something didn't work, the Linux source code was available. Our most important system was our database, used to process orders, so to play it safe we kept that on the original Unix SVR2 box.

Using dumb serial communications boards, we connected the terminals to the Linux system and used `rlogin` to access the database over the TCP/IP network. With Linux up and running reliably, we could then add a continuously available Internet connection. Linux integrated into our environment easily and proved to be very stable. After a move to a bigger office in 1994, we started replacing the remaining dumb terminals with Linux systems, many based on older 386 and 486 machines. While MS-Windows required more horsepower, Linux could bring life back into these older systems.

Supporting the Web

As our company grew, so did our need to create an Internet presence. We needed both a Web server and a firewall to isolate our internal network from the Internet. Linux met both of these needs.

Our first Web server was a 486/100 with 16 Mbytes of RAM. It ran Apache, a popular open-source Web server that is included with Linux. We decided that if the server received 10,000 hits per day, we would then see what additional hardware we needed to support future growth. That 486/100-based Web server performed well up to over 100,000 hits per day! At that point we finally upgraded it to an AMD K5 processor, added more RAM, and increased the disk capacity.

Today, our Web server handles 200,000 to 300,000 hits per day for our four different Internet domains: www.ssc.com, www.linuxjournal.com, www.linuxgazette.com, and www.linuxresources.com. We have added a secure server and an

Continued on the next page

look like relatively "dumb" client interfaces that may at first do little more than collect and present data, with the majority of the data processing taking place at some central (often legacy) processing site.

The first stage in RID is to place low-cost but powerful PC resources at the remote sites, ensuring that the PCs have sufficient "excess capacity" to accommodate expansion to the full set of desired features. While the hardware installation is being planned, the minimal set of critical requirements is implemented as simply and quickly as possible, generally by using high-level scripting and shell tools rather than compiled languages, and deployed to the target sites at about the same time as the hardware. Once this initial deployment is complete, operations using the minimal set of software functions can begin immediately.

Once a RID site goes "live" in this way, the process of updating and changing the software is handled remotely (for example, over the Internet) instead of through on-site personnel visits. When many different sites are involved, or when the sites are unusually inaccessible, this remote-update ability alone can represent a substantial savings to the overall development program.

A RID strategy depends critically on the ability of the OS to remain stable while changes are being made to its applications and, in some cases, to the OS kernel itself. Without a high level of kernel stability, the strategy becomes unworkable; frequent system failures would force costly on-site maintenance visits and support. On the other hand, an OS that is unaffected by such changes allows updates to be handled by simple downloading.

Bring In the Supercomputers

The application example by Becker, Ligon, Merkey, and Ross demonstrates the unique advantage of having both access to source code and an incentive to use it. The NASA Beowulf Project was

assortment of CGI scripts, mostly written in Perl, and will soon add a third server.

With Linux, the firewall also proved easy to implement. Linux includes firewall code, so we simply had to configure a Linux system with two Ethernet cards to act as a firewall, and set it up between the internal network and our externally visible systems.

Linux Stability

While making all these changes, we still had our database running on the Unix SVR2 system. Although it had been reliable enough, we wanted to replace it with Linux as well. We tried unsuccessfully to convince Progress Software, the database's maker, to port to Linux. Our alternate plan was to get the SCO Unix version of Progress and run it under iBCS, an emulator package that permits running other Intel Unix binaries on Linux.

The combination of an SCO Unix version of the Progress database on top of iBCS and Linux was fairly easy to get running, and has since proven extremely stable. Uptimes on the database machine are generally measured in hundreds of days and performance has been excellent. (For more details, see "Linux Means Business: Using Progress at SSC" by Peter Struijk and Lydia Kinata, *Linux Journal*, Sept. 1996, http://www.linuxjournal.com/issue29/SSC.html.)

We have also found Samba, a tool that comes with Linux, useful for cross-platform work. Samba allows Linux to emulate a file and print server for Microsoft clients. It is totally transparent to the Microsoft clients, and offers the stability Linux users expect. Samba can also be used to accept PostScript printouts from a Windows 95 system, then print them on a non-PostScript Linux printer by running them through a Linux utility called GhostScript.

Today's Configuration

Today, all our desktops run X Windows, and most have been upgraded to more powerful CPUs such as an AMD K5 or K6. We have been using StarOffice (www.stardivision.com), Applixware (www.applix.com), and WordPerfect—all are licensed software, and each has its followers. Within the next six months, we will probably pick one as the office standard. Netscape is the usual choice for Web browsing, and elm is used for most e-mail.

Virtually all our systems are now running Debian Linux (www.debian.org). Although not the most popular distribution, it has been a great fit for us. Debian's update mechanisms permit upgrading systems with little (if any) downtime.

Our progression from a two-person company running on dumb terminals connected to a Unix box to a 15-person company running on Linux workstations has been more evolution than revolution. While we do not consistently pick Linux, we always look at it first to see if it is an option.

The Future

The obvious future for us would be a 100-percent Linux office. Linux has proven so reliable and easy to maintain that converting the remaining systems to Linux seems an obvious choice. Is this likely? In the short term, no. To be 100-percent Linux, we would need to have QuarkXPress, Adobe Photoshop, FrameMaker, and CorelDraw all running on Linux. Corel is on the way (www.corel.com), and we could consider an alternative to FrameMaker such as Ventura Publisher. This leaves Quark and Photoshop as our major roadblocks.

Being Linux users as well as a publisher of Linux products, we have an obvious interest in talking to vendors about the advantages of porting products to Linux. As more companies start using Linux, it should become easier to convince vendors that Linux is a viable home for their products. ❖

Phil Hughes is cofounder of SSC, a publishing company specializing in Linux and Unix documentation. Readers may contact him at fyl@ssc.com.

an effort to create parallel-processing supercomputers out of ordinary, inexpensive PCs. Initially, the developers were constrained by the lack of a sufficiently fast way to let the PCs communicate with each other. By taking advantage of the source code drivers in Linux, one of the authors (Becker) was able to create a new type of "channel bonded" Ethernet connection and driver that allowed ordinary Ethernet connections to be used in parallel. This result was folded back into the Linux community, and consequently Linux now provides one of the best ways available for cash-strapped research organizations around the world to create low-cost parallel-processing supercomputers. The efficiency of Linux, which many users describe as "making a 486 look like a Pentium," also contributes to its use in such supercomputers.

Mission-Critical Defense Applications

The US Department of Defense and its affiliates have also noticed the low cost, high reliability, versatility, and rapid deployment of Linux. For example, Frank McPherson describes how his group in The Mitre Corporation is building a prototype that will evaluate the possible use of Linux in tanks and other military vehicles. Other applications of Linux also exist within the DoD, such as in servers where reliability and security are paramount.

One of the ironies of Linux for defense work is its appropriateness for high-security applications. Peer review, self-interest, and source code availability are the reasons why. Linux has been

GNU/Linux Enables a High-Volume, Soft Real-Time Application

Jorge Ocón, INDRA

I work for the Control Systems and Automation Division of the INDRA Corporation, a Spanish company that employs more than 3,000 information technology professionals. Our division develops real-time control systems for electrical facilities, traffic systems, and mass transportation systems such as railways.

A friend introduced me to GNU/Linux in 1995 by giving me a CD-ROM copy of it. Initially my colleagues and I viewed GNU/Linux simply as a way to learn Unix at home. However, we soon realized that GNU/Linux was a robust, reliable system that surpassed the performance of many commercial systems. In 1996, I introduced GNU/Linux in my department as a way to reduce the load on our Unix servers. This allowed developers to verify GNU/Linux reliability for themselves, and to benchmark it against other operating systems.

Our first use of GNU/Linux in a production system was for a project in Argentina with Buenos Aires Railways, in which we used it as the operating system for desktop ticket dispensers and station hubs. Desktop ticket dispensers must be highly reliable, immune to fraud, and easy to use. Station hubs are embedded systems that control and communicate with other station machines such as turnstiles and vending machines. Station hubs collect passenger access and sales data and store it to disk for later transfer to a system-wide database. The ticket dispensers and station hubs are examples of "soft real-time" systems—they must send and receive messages within 100 to 200 milliseconds.

Our initial task was to design and install 12 ticket dispensers and associated hubs in the ONCE rail station in Buenos Aires. This will later be expanded to include all 41 Buenos Aires Railways stations with 98 ticket dispensers and 92 station hubs, which collectively handle half a million passengers per day. Our equipment must operate 24 hours a day, seven days a week, and would ideally be shut down only for hardware maintenance or software reconfiguration.

It took six months to develop the station hubs and ticket dispensers in Spain, then we shipped everything to Argentina, where we spent three months installing them, teaching our clients how to use them, and fixing problems. The whole project took about a year and a half and involved about 20 developers—not many considering we developed, manufactured, and installed the turnstiles, vending machines, station hubs, ticket dispensers, control center (which included a Sun Server 3000), and communication systems for the stations.

I considered using MS-DOS/Windows, GNU/Linux, and other PC-based versions of Unix, but eventually decided to use GNU/Linux, in main part because of its network capabilities: native TCP/IP, NFS, and Ethernet are included with it. GNU/Linux also efficiently uses resources, so that a machine with just 8 Mbytes of RAM could run many tasks simultaneously. The ease with which we could modify GNU/Linux allowed us to increase efficiency by launching only required applications, develop our own communication drivers for use with custom cards, and simplify field installation by installing the operating system and all our applications in a single pass. Another advantage was that GNU/Linux boxes were able to manage their own resources automatically. This minimized software maintenance and made the systems less vulnerable to attack. Finally, the availability of source code in GNU/Linux allowed us to develop our systems more rapidly.

The GNU software we are using includes the ncurses library; utilities to develop module device drivers (insmod, rmmod, lsmod); image manipulation tools (xpaint, display); NFS; PPP drivers; X servers for several video adapters; powerd for the UPS; Ethernet drivers for SMC, 3COM, and DEC boards; dialog libraries; mtools; and the well-known standards make, tar, and gzip. The C compiler gcc, the emacs and jed editors, and the GNU debuggers (gdb and xxgdb) have proven to be excellent.

The main disadvantage of GNU/Linux was its lack of traditional support, which we solved by creating a skilled team consisting of Antonio Benito, Berta Madrid, and Roberto Peña. Their expertise in both GNU/Linux and transportation systems assured the success of the Buenos Aires Railways project.

Based both on our earlier experiences with GNU/Linux at INDRA and the success of our initial installation of ticket dispensers and station hubs for Buenos Aires Railways, I am convinced that open-source software has already reached a level of maturity appropriate for professional use in industrial projects. Open-source software also presents advantages over commercial software that go well beyond low cost, such as the flexibility and adaptability provided by having the source code available. I believe that, like the Internet, open-source software is a new phenomenon with unstoppable vitality. In particular, open-source software makes possible the free communication of ideas within the international community, and so benefits everyone involved. ❖

Jorge Ocón is a software engineer at INDRA Corporation in Spain, where he develops real-time control systems for electrical facilities, traffic systems, and mass transportation systems. Readers may contact him at joa@jet.es.

Linux Provides Reliability for Critical Oceanographic Monitoring

Jon Ashley, Proudman Oceanographic Laboratory

The Proudman Oceanographic Laboratory, part of the UK's Centre for Coastal and Marine Sciences, has its own team of engineers who work closely with the scientists to develop new instruments for monitoring ocean parameters. One of our main projects is measuring tidal information from a network of about 50 tide gauges around the UK coast, at remote island and coastal locations worldwide (including Antarctica), and even on the ocean floor. The needs of our small team of engineers are fairly specialized; foremost among them is reliability. With many gauges at remote and inaccessible locations worldwide, we simply cannot afford costly on-site gauge repairs and servicing.

We began using Linux when an effort to replace an earlier network of simple gauges with a new, feature-packed system fell behind schedule. The new network was to provide more environmental sensors, continuous monitoring, and easier access to data. Instead of following the original plan of using custom hardware, we decided to install Linux on a PC with no monitor or keyboard. We modified various pieces of front-end sensors to output data in serial format, then connected them to the PC. The initial software for logging data was extremely simple, consisting simply of built-in Linux multitasking features that ran low-level "cat" (concatenation) I/O processes to copy data from serial ports into files. Adding new sensors thus became just a matter of adding serial ports.

This simple approach proved acceptable because Linux permitted us to download and install new software remotely over the telephone link to the gauge. This meant that even if the first application did little more than collect the data, we could build up its capability over time—without ever having to visit the gauge site.

Within two weeks, we had the first prototype running at the UK site of Holyhead. The original plan supported nontechnical users' access to data through a front-end client program running under Microsoft Windows. With Linux, we simply sent a copy of the Apache Web server to the new tide gauge and set up a Web page. A few short, simple CGI scripts later, we were able to dial up the system, start a Web browser, and be greeted with an attractive page welcoming us to the CCMS Tide Gauge at Holyhead. We could select parameters and view graphs of the latest data, generated entirely on demand. The system was already outperforming the original specifications!

The new system was reliable enough that when it came time to service the tide gauge at the remote island site of St. Helena, we decided to install it there. We had to be completely confident of Linux's stability, since it would be difficult or even impossible to dispatch someone to service or reboot the system at this site.

So far, the installation at St. Helena has worked splendidly. The only major problem we have encountered was a main power outage that lasted longer than the backup batteries could support. In other areas we are only gradually realizing the advantages the new system gives us. For example, downloading raw data from St. Helena over telephone lines is very expensive, but we soon realized that we could send our data-processing software over to the tide gauge and get it to process the data at the gauge before transmitting the much smaller results.

For applications such as ours, Linux provides several powerful advantages, among them high reliability, low cost (both for software and hardware), very fast prototyping times, proper multitasking, and excellent remote control. From an engineer's point of view, Linux provides excellent all-around benefits beyond access to source code, including good hardware access, a wide range of software tools, Unix compatibility, and good documentation and support. ❖

Jon Ashley is an electronic engineer specializing in embedded systems, currently designing deep-water instrumentation for the Proudman Oceanographic Laboratory. He has been using Linux for around three years. Readers may contact him at joa@ccms.ac.uk.

undergoing intensive peer review for many years by an unusually large and experienced international cadre of software developers, which contrasts sharply with the limited and sporadic code review resources of most commercial software organizations. Because Linux tends to be the primary development and support platform of these reviewers, their search for such holes tends to be strongly motivated by self-interest and pride in their contributions to Linux. As in any kind of peer review, such direct, personal motivations tend to result in more effective and insightful code reviews. Finally, the availability of Linux source code makes any long-term "hiding" of poorly designed or even malicious software within binary modules essentially impossible. This is in sharp contrast to closed-source software, where spurious code such as "Easter Eggs" (unlisted binary executables with

Linux for Integrated and Embedded Military Applications

Frank McPherson, The Mitre Corporation

Mitre's Embedded Hardware Systems Specialty Group is developing a prototype embedded system based on the Linux operating system and a commercial off-the-shelf VMEbus card. The system potentially will be used in US Army weapons systems such as the Abrams main battle tank, the Bradley armored personnel carrier, the Kiowa scout helicopter, and the Apache Longbow attack helicopter.

The Army uses real-time embedded systems to control mission-critical operations such as turning a turret or firing a gun. Lately, however, as it works towards building the First Digitized Division, it has expanded its view of integrated computing to include more complex applications, such as situational awareness, that require computers with desktop-like displays and input devices. Because the design goals and requirements of these applications, and thus of the systems used to host them, differ from those of real-time embedded systems, we recommend partitioning the software by placing these applications on a separate integrated computer.

The prototype hardware is the DY4-177 VME single-board computer built around the Motorola PowerPC 603e, including an on-board PCI bus and COTS chipsets for SCSI, serial, and Ethernet support. We chose the PowerPC for the prototype effort because it is available in a rugged version that meets stringent military environmental requirements, including the ability to operate in high-temperature and high-shock environments such as those encountered by the Abrams battle tank.

Mitre is developing a boot loader for Linux on the DY4-177 board, and is modifying existing Linux driver software to support the SCSI, serial, and Ethernet hardware on the board. Mitre will complete these tasks in April 1999 and plans to open-source the boot loader and driver modifications. ❖

Frank McPherson is a software engineer and systems integrator for the MITRE Corporation. His recent projects include porting Linux to the DY4-177 VME board and providing technical support for the architecture and integration of Embedded Battle Command situational awareness software to PM Abrams. Readers may contact him at frank@mitre.org.

Using Linux To Build Fast Networks

Erann Gat, Jet Propulsion Laboratory
Mike Ciholas, Cedar Technologies

We are using Linux to develop a high-speed network called FlowNet. This project has been a "virtual garage" operation, involving one person in California and one at various times in Boston, Pennsylvania, and Indiana. Working alone in our spare time, we have developed FlowNet on a shoestring budget—about $20,000 for a dozen prototypes.

FlowNet integrates ATM and Ethernet features and is a speed-scalable network that supports both packet-oriented and connection-oriented traffic. FlowNet also provides quality-of-service guarantees at layer 2 of the OSI network service model. FlowNet operates on standard Cat 5 copper cable to provide 500 Mbps service.

Open-source software, including Linux and Intel's gnu960 development tools, was instrumental to FlowNet's development. We used Linux to develop both the on-board firmware and the device drivers for FlowNet. Several Linux features proved crucial to meeting our objectives. The first was the availability of model code for device drivers. Because the FlowNet interface resembles Ethernet, we were able to use Donald Becker's Tulip driver as a model and adapt it for FlowNet rather than starting from scratch.

The second Linux feature that helped immeasurably was kernel modules. Because device driver code is kernel code, we could not run it as an application. Without modules, device drivers must be tested by compiling them into the kernel and rebooting. This can extend the development cycle by many minutes. Kernel modules permit dynamic linking and unlinking of kernel code, reducing the testing cycle to less than a minute. We built a kernel module for FlowNet that loaded the card's firmware through the PCI bus during initialization. This made it possible to recompile and restart all the FlowNet software with a single `make` command. Consequently, we developed all the software for FlowNet in less than three months.

We would not have been able to develop FlowNet without Linux as a development platform. The hardware costs stretched our meager budget to the limit; the development tools needed to develop FlowNet for a commercial OS would have killed the project. Linux made it possible to build FlowNet privately—it almost certainly could not have happened any other way. ❖

Erann Gat is a senior member of the technical staff at the NASA Jet Propulsion Laboratory in Pasadena, California. **Mike Ciholas** is the founder of Cedar Technologies, an electronic hardware design contractor in Newburgh, Indiana. Contact the authors at {gat, mikec}@flownet.com.

BEOWULF: LOW-COST SUPERCOMPUTING USING LINUX

Donald Becker, Walter Ligon, Phillip Merkey, and Robert Ross

A *Beowulf cluster computer* is a low-cost supercomputer with three dedicated components: a collection of low-cost computers (generally PCs) of a single type, a high-speed network for interconnecting them, and an operating system (Linux) that lets the computers operate efficiently together as a parallel system. Thomas Sterling and Don Becker constructed the first Beowulf computer in 1994 at the Goddard Space Flight Center in Greenbelt, Maryland, using a cluster of 16 off-the-shelf 486 processors. Sterling and Becker were working for the Center for Excellence in Space Data and Information Sciences (CESDIS), and their work was sponsored by the Earth and Space Sciences (ESS), whose overall goal was to extend the boundaries of scientific computing into the teraflops range.

Although their use of low-cost PCs makes Beowulf computers superficially similar to Internet-based computing, in which many "volunteer" PCs or systems are interconnected via the Internet, Beowulfs differ in their use of dedicated high-speed communication links that let them solve a much broader range of problems. In general, Internet-based computing works best for problems that can be partitioned into many small, independent tasks that interact little during computation. It does not work well for problems such as weather simulation, which requires fast, extensive communication between individual processing units. A Beowulf computer can better handle this type of problem because it is capable of rapid and highly synchronized internal communication between processors.

Beowulfs are possible because low-cost "commodity" PCs can be purchased in large quantities and fast networks are available to interconnect them. The Linux operating system's unique properties also helps make Beowulfs possible—only Linux currently provides the right combination of low cost, reliability, source-code modifiability, and driver support needed for a collection of PCs work together as an effective, low-cost supercomputer. The first Beowulf cluster computer was interconnected using a "channel-bonded" Ethernet technology developed by Don Becker specifically for use in the Beowulf effort.

Linux software has also contributed to Beowulf through its maturity, robustness, and long-term stability. Historically, the need for unique or custom system software in high-performance computing has resulted in immature system software that changed over time. For Beowulf, Linux and GNU software provide a ready-made level of maturity that far exceeds that of most custom high-performance system software, and also gives programmers a familiar Unix-style environment. Linux also supports communication standards for parallel computers, such as the Parallel Virtual Machine standard for creating highly concurrent applications and the Message Passing Interface standard for message-passing parallel programs.

The Sterling and Becker concept of using low-cost, off-the-shelf commercial hardware to build powerful parallel computers was an immediate success, and it quickly spread through NASA, into the academic and research communities, and around the world. Beowulf computers are especially attractive for groups that need supercomputing but would prefer to avoid the high cost of a fully custom parallel computer. Rapid drops in PC prices make it likely that Beowulf computers will proliferate in the next few years, particularly in developing countries. The spread of Beowulf is also assured because its software environment is implemented as add-ons to royalty-free basic distributions of Linux.

The future of the Beowulf systems looks promising. As microprocessor technology continues to progress, higher-speed networks become cost-effective, and more application developers move to parallel computing platforms, Beowulf systems will likely continue to expand and help fill the internationally important niche of low-cost supercomputing. ❖

Donald Becker is a staff scientist and **Phillip Merkey** is a senior scientist at the Universities Space Research Association, Center of Excellence in Space Data and Information Sciences at Goddard Space Flight Center. **Walter Ligon** is a professor of computer engineering and **Robert Ross** is a PhD candidate in computer architecture at Clemson University. Contact Ligon by e-mail at walt@eng.clemson.edu.

generally innocuous, often humorous intent) has been found in the released versions of some of the most popular desktop tools.

LOVE THOSE KERNELS FROM LINUX

Finally, the application from Erann Gat and Mike Ciholas demonstrates the use of a feature of Linux called kernel modules, which provide an example of the unusual degree of configurability of Linux. Kernel modules allow new device drivers to be dynamically linked to and unlinked from an active Linux kernel, greatly reducing the time and trouble needed to test novel driver code. Like the Beowulf example, this application also demonstrates how the open-source features of Linux benefit users trying to extend the limits of existing hardware. ❖

Terry Bollinger is a principal information systems engineer at The Mitre Corp., where he focuses on distributed software architectures. Readers may contact him at terry@mitre.org.