Advanced search

Linux Journal Issue #109/May 2003

GUI security management: Firewall Builder
LINUX JOURNAL
Software Synth roundup
Kernel Developer Robert Love
Are You Ready to Rock?
Special 2.6 Kernel Preview:
Killer Audio Features and more
Easy web content management with Plone

Kernel Mode Linux

Toshiyuki Maeda

Issue #109, May 2003

Now you don't have to write a module to run a program in kernel space. Run any program there with this patch.

Kernel Mode Linux (KML) is a technology that enables the execution of ordinary user-space programs inside kernel space. This article presents the background, an approach and an implementation of KML. A brief performance experiment also is presented.

Traditional kernels protect themselves by using the hardware facilities of CPUs. For example, the Linux kernel protects itself by using a CPU's privilege-level facility and memory protection facility. The kernel assigns itself the most-privileged level, kernel mode. User processes are at the least-privileged level, user mode. Thus, the kernel is protected by CPUs, because programs executed in user mode cannot access memory that belongs to programs executed in kernel mode.

This protection-by-hardware approach, however, has a problem: user processes cannot access the kernel completely. That is, the kernel cannot provide any useful services, such as filesystems, network communication and process management, to user processes. In short, user processes cannot invoke system calls in the kernel.

To cope with this problem, traditional kernels exploit hardware facilities that modern CPUs provide for, escalating a program's privilege level in a safe and restricted way. For example, the Linux kernel for the IA-32 platform uses a software interrupt mechanism inherent to IA-32. The software interrupt can be seen as a special jump instruction whose target address is restricted by the kernel. At initialization, the kernel sets the target address of the software interrupt to the address of a special routine that handles system calls. To invoke system calls, a user program executes a special instruction, int 0x80. Then, the system-call handling routine in the kernel is executed in kernel mode.

The routine performs a context switch; that is, it saves the content of the registers of the user program. Finally, it calls the kernel function that implements the system service specified by the user program.

The system call-by-hardware approach can become very slow, however, because the software interrupt and the context switch require heavy and complex operations. On the recent Pentium 4, the software interrupt and context switch is about 132 times slower than a mere function call.

By the way, recent Linux kernels for IA-32, versions 2.5.53 and later, use a pair of special instructions, sysenter and sysexit, for system calls. But, this is still about 36 times slower than a mere function call.

The obvious way to accelerate system calls is to execute user processes in kernel mode. Then, system calls are handled quickly because no software interrupts and context switches are needed. They can be function calls only, because the user processes can access the kernel directly. This approach may seem to have a security problem, because the user processes executed in kernel mode can access arbitrary portions of the kernel. Recent advances in static program analysis, such as type theory, can be used to protect the kernel from user processes. Many technologies enable this protection-by-software approach, including Java bytecode, .NET CIL, O'Caml, Typed Assembly Language and Proof-Carrying Code.

KML: Execute User Processes in Kernel Mode

As a first step toward a kernel protected by software, I have implemented KML. KML is a modified Linux kernel that executes user processes in kernel mode, which then are called kernel-mode user processes. Kernel-mode user processes can interact with the kernel directly. Therefore, the overhead of system calls can be eliminated.

KML is provided as a patch to the source of the original Linux kernel, so you need to build the kernel from the source. To use KML, apply the patch and enable Kernel Mode Linux when you configure your kernel. Build and install the kernel, and then reboot. The KML patch is available from www.yl.is.s.u-tokyo.ac.jp/~tosh/kml.

In current KML, programs under the directory /trusted are run as kernel-mode user processes. The kernel itself doesn't perform any safety check. For example, the following commands:

```
  % cp /bin/bash /trusted/bin && /trusted/bin/bash
```

execute bash in kernel mode.

What Kernel-Mode User Processes Can Do

Kernel-mode user processes are ordinary user processes except, of course, for their privilege level. Therefore, they basically can do whatever an ordinary user process can do. For example, a kernel-mode user process can invoke all system calls, even fork, clone and mmap. In addition, if you use a recent GNU C library (2.3.2 and later or the development version from CVS), system calls are translated automatically to function calls in kernel-mode user processes, with a few exceptions, such as clone. Therefore, the overhead of system calls in your program is removed without modifying it.

The paging mechanism also works. That is, kernel-mode user processes each have their own address space, the same as ordinary user processes. Moreover, even if the kernel-mode user process excessively allocates huge memory, the kernel automatically pages out the memory, as it does for ordinary user processes.

Exceptions, such as segmentation faults and illegal instruction exceptions, can be handled the same as an ordinary user process, unless the program improperly accesses the memory of the kernel or improperly executes privileged instructions. As an example, build the following program and execute it as a kernel-mode process:

```
int main(int argc, char* argv[])
{
    *(int*)0 = 1;
    return 0;
}
```

The process is terminated by a segmentation fault exception, without a kernel panic. This example also indicates that the signal mechanism works.

As a second example, build the following program and execute it as a kernel-mode user process:

```
int main(int argc, char* argv[])
{
    for (;;);
    return 0;
}
```

Then, use Ctrl-C to send SIGINT to the process. Notice that it receives the signal and exits normally.

This second example also indicates that process scheduling works. That is, even if a kernel-mode user process enters an infinite loop, the kernel preempts the process and executes other processes. You may have noticed already that your system did not hang, even in the infinite loop of this example.

What Kernel-Mode User Processes Cannot Do

Although kernel-mode user processes are ordinary user processes, they have a few limitations. If a kernel-mode user process violates these limitations, the system will be in an undefined state. In the worst-case scenario, your system may be broken.

Limitation 1: don't modify the CS, DS, SS or FS segment register. Current KML for IA-32 assumes that these segment registers are not modified by kernel-mode user processes, and it uses them internally.

Limitation 2: don't perform privileged actions improperly. In kernel mode, programs can perform any privileged action. However, if your program performs such actions in a way that is inconsistent with the kernel, the system will be in an undefined state. For example, if you execute the following program as a kernel-mode user process:

```
int main(int argc, char* argv[])
{
        /* disable hardware interrupts */
        __asm__ __volatile__ ("cli");
        for (;;);
        return 0;
}
```

your system will hang.

In my experience, few applications violate these limitations. Ones that do violate them include WINE and VMware. These limitations are against only kernel-mode user processes. Ordinary user processes are never affected by these limitations, even when running on a KML-capable kernel.

KML Internals

In IA-32 CPUs, the privilege level of an executed program is determined by the privilege level of the code segment in which the program is executed. Recall that a program counter for IA-32 CPUs consists of a segment, specified by the CS segment register, and an offset into the segment, the EIP register. The privilege level of the code segment then is determined by its segment descriptor. A segment descriptor has a field for specifying the privilege level of the segment.

Basically, the Linux kernel prepares two segments, the kernel code segment and the user code segment. The kernel code segment is used for the kernel itself, and its privilege level is kernel mode. The user code segment is used for ordinary user processes, and its privilege level is user mode. When using execve on a user process, the original Linux kernel sets its CS segment register to the user code segment. Thus, the user process is executed in user mode.

To execute a user process as a kernel-mode user process, the only thing KML does is set the CS register of the process to the kernel code segment, instead of to the user code segment. Then the process is executed in kernel mode. Because of KML's simple approach, a kernel-mode user process can be an ordinary user process.

The Stack Starvation Problem and Its Solution

As described in the previous section, the basic approach of KML is quite simple. Its big problem is called stack starvation. First, I'll explain how the original Linux kernel handles exceptions (page faults) and interrupts (timer interrupts) on IA-32 CPUs. Then, I'll describe the stack starvation problem. Finally, I'll present my solution to the problem.

In the original Linux kernel, interrupts are handled by interrupt handling routines specified as gates in the Interrupt Descriptor Table (IDT). When an interrupt occurs, an IA-32 CPU stops execution of the running program, saves the execution context of the program and executes the interrupt handling routine.

How the IA-32 CPU saves the execution context of a running program at interrupts depends on the privilege level of the program. If the program is executed in user mode, the IA-32 CPU automatically switches its memory stack to a kernel stack. Then, it saves the execution context (EIP, CS, EFLAGS, ESP and SS register) to the kernel stack. On the other hand, if the program is executed in kernel mode, the IA-32 CPU doesn't switch its memory stack and saves the context (EIP, CS and EFLAGS register) to the memory stack of the running program.

What happens if a kernel-mode user process of KML accesses its memory stack, which is not mapped by the page tables of a CPU? First, a page fault occurs, and the CPU tries to interrupt the process and jump to a page fault handler specified in the IDT. However, the CPU can't accomplish this work, because there is no stack for saving the execution context. Because the process is executed in kernel mode, the CPU can never switch the memory stack to the kernel stack. To signal this fatal situation, the CPU tries to generate a special exception, a double fault. Again, the CPU can't generate the double fault, because there is no stack for saving the execution context of the running process. Finally, the CPU gives up and resets itself.

To solve this stack starvation problem, KML exploits the task management facility of IA-32 CPUs. The IA-32 task management facility is provided to support process management for kernels. Using the facility, a kernel can switch between processes with only one instruction. However, today's kernels don't use this facility, because it is slower than software-only approaches. Thus, the facility is almost forgotten by all.

The strength of this task management facility in IA-32 CPUs is that it can be used to handle interrupts and exceptions. Tasks managed by an IA-32 CPU can be set to the IDT. If an interrupt occurs and a task is assigned to handle the interrupt, the CPU first saves the execution context of the interrupted program to a task data structure of the program instead of to the memory stacks. Then, the CPU restores the context from the task data structure specified in the IDT.

The most important point is there is no need to switch a memory stack if the task management facility is used to handle interrupts. That is, if we handle page fault exceptions with the facility, a kernel-mode user process can access its memory stack safely.

However, if we handle all page faults with the facility, the performance of the whole system degrades, because the task-based interrupt handling is slower than the ordinary interrupt handling.

Therefore, we handle only double fault exceptions this way. So, only page faults caused by memory stack absence are handled by the task management facility. In my experience, memory stacks rarely cause page faults, and the performance decrement is negligible.

Performance Measurement

To measure the degree of performance improvement, I conducted two experiments. Both experiments compared performance of the original Linux kernel and KML. I used the sysenter/sysexit mechanism for performance measurement of the original Linux kernel, instead of the int 0x80 instruction. The experimental environment is shown in Table 1.

<u>Table 1. Experimental Environment</u>

In the first experiment, I measured the latency of the getpid and gettimeofday system calls. In the measurement, the system calls were invoked directly by user programs, without libc. The latency was measured with the rdtsc instruction. The result is shown in Table 2.

<u>Table 2. Latency of System Calls (Unit: CPU Cycles)</u>

The result shows that getpid was 36 times faster in KML than in the original Linux kernel, and gettimeofday was twice as fast in KML as it was in the original Linux kernel.

The second experiment is a file I/O benchmark using the Iozone filesystem benchmark. I measured the throughput of four types of file I/O: write, rewrite, read and reread. The measurements were performed on various file sizes from 16KB to 256KB, and the buffer size was fixed at 8KB. The underlying filesystem was ext3. In each measurement, I executed the Iozone benchmark 30 times and chose the best throughput.

The throughput of reread is shown in Table 3. Due to space limitations, the detailed results for write, rewrite and read are omitted.

Table 3. Throughput of reread: Buffer Size=8KB

The result shows that the throughput of reread in KML was improved by 6.8-21%. In addition, write was improved by 0.6-3.2%, rewrite was improved by 3.3-5.3% and read was improved by 3.1-15%.

These experimental results indicate that KML can improve the performance of applications that invoke system calls often, such as those that read or write many small files. For example, web servers and databases can be executed efficiently in KML.

I've performed a benchmark for the Apache HTTP server on KML. It didn't show performance improvement, because I have only a 100-base Ethernet LAN, which became the main bottleneck. If I perform the benchmark on a faster network (say, 1000-base Ethernet or faster), I predict it will show performance improvement.

In the preceeding experiments, it is worth noting that KML eliminated only the overhead of system calls. With some modification to the application, KML will be able to do more for performance improvement. For example, kernel-mode user processes can access I/O buffers directly in the kernel to improve I/O performance.

Resources

email: tosh@is.s.u-tokyo.ac.jp

Toshiyuki Maeda is a PhD candidate in Computer Science at the University of Tokyo. His favorite comics are Hikaru no GO (*Hikaru's Go*), *Jojo no Kimyo na*

Boken (*Jojo's Bizarre Adventure*) and *Runatikku Zatsugidan* (*Lunatic Acrobatic Troupe*).

Introducing the 2.6 Kernel

Robert Love

Issue #109, May 2003

Scheduler and audio improvements are only two of the features you'll notice when the 2.5 development series becomes 2.6. Here's a kernel hacker's view into the near future.

The kernel has come a long way since Linus branched off 2.4.15 to create 2.5.0 back on November 22, 2001. Since then, rapid development has ensued, resulting in a drastically different and much-improved kernel. This article discusses the more interesting, important features and their impact on the performance and reliability of Linux.

History of 2.5 Thus Far

In Linux kernel parlance, the minor version number of a kernel denotes whether that kernel belongs to a stable series or a development series. Even minor version numbers denote stable kernels, and odd minor version numbers denote development kernels. When a development kernel is fully mature and deemed stable, its minor version number is incremented to an even value. For example, the 2.3 kernel development series gave way to the 2.4 stable series.

The current development kernel is 2.5. The initial work on a development series is quite brisk, and many new features and improvements are incorporated. When Linus and the kernel developers are satisfied with the new feature set, a feature-freeze is declared, which has the purpose of slowing development. The last feature-freeze occurred on October 31, 2002. Ideally, when a feature-freeze is declared, Linus will not accept new features—only additions to existing work. When the existing features are complete and nearly stable, a code-freeze is declared. During a code-freeze, only bug fixes are accepted, in order to prepare the kernel for a stable release.

When the development series is complete, Linus releases the kernel as stable. This time around, the stable kernel most likely will be version 2.6.0. Although the official release date is “when it is done”, a good estimate is third or fourth quarter 2003.

In March 2001 and again in June 2002, the core kernel developers met at Kernel Summits to discuss the kernel. The primary goal of 2.5 was to bring the aging block layer (the part of the kernel responsible for block devices, such as hard drives) into the 21st century. Other concerns centered on scalability, system response and virtual memory (VM). The kernel hackers met all—and many more—of these goals. The list of important new features includes:

- O(1) scheduler
- preemptive kernel
- latency improvements
- redesigned block layer
- improved VM subsystem
- improved threading support
- new sound layer

In this article, I discuss a lot of new technology and innovation that has gone into the 2.5 kernel and will appear in 2.6. The development is the result of hard work from many individuals. I am going to refrain from mentioning names, because if I start giving credit I inevitably will miss some people, and I would rather give no list than an incomplete or incorrect one. The Linux Kernel Mailing List archive is a good source of who did what.

O(1) Scheduler

The process scheduler (or, simply, the scheduler) is the subsystem of the kernel responsible for allocating processor time. It decides which process gets to run when. This is not always an easy job. From a possibly large list of processes, the scheduler must ensure that the most worthy one is always running. When there is a large number of runnable processes, selecting the best process may take some time. Machines with multiple processors only add to the challenge.

Improvements to the scheduler ranked high on the list of needed improvements. Specifically, developers had three specific goals:

- The scheduler should provide full O(1) scheduling. Every algorithm in the scheduler should complete in constant time, regardless of the number of running processes.
- The scheduler should have perfect SMP scalability. Ideally, each processor should have its own locking and individual runqueue. A runqueue is the list of runnable processes from which the scheduler chooses.

- The scheduler should have improved SMP affinity. It should naturally attempt to group tasks on a specific CPU and run them there. It should migrate tasks from one CPU to another only to resolve imbalances in runqueue length.

The new scheduler accomplishes all of these goals. The first goal was full O(1) scheduling. O(1) denotes an algorithm that executes in constant (fixed) time. The number of runnable tasks on a system—or any other variable, for that matter—has no bearing on the time to execute any part of the scheduler. Consider the algorithm for deciding which task should run next. This job involves looking at the highest priority task, with timeslice remaining, that is runnable. In the previous scheduler, the algorithm was analogous to:

```
for (each runnable process on the system) {
        find worthiness of this process
        if (this is the worthiest process yet) {
                remember it
        }
}
run the most worthy process
```

With this algorithm, the worthiness of each process must be checked. This implies the algorithm loops *n*-times for *n* processes. Hence, this is an O(*n*) algorithm—it scales linearly with the number of processes.

Conversely, the new scheduler is constant with respect to the number of processes; it does not matter whether there are five or 5,000 runnable processes on the system. It always takes the same amount of time to select and begin executing a new process:

```
get the highest priority level that has processes
get first process in the list at that priority level
run this process
```

In this algorithm, it is possible to simply “get the highest priority level” and “get first process in the list”, because the scheduler keeps track of these things. It simply has to look up, instead of search for, these values. Consequently, the new scheduler can select the next process to schedule without looping over all runnable processes.

The second goal is perfect SMP scalability. This implies the performance of the scheduler on a given processor remains the same as one adds more processors to the system, which was not the case with the previous scheduler. Instead, the performance of the scheduler degraded as the number of processors increased, due to lock contention. The overhead of keeping the scheduler and all of its data structures consistent is reasonably high, and the largest source of this contention was the runqueue. To ensure that only one processor can concurrently manipulate the runqueue, it is protected by a lock. This means, effectively, only one processor can execute the scheduler concurrently.

To solve this problem, the new scheduler divides the single global runqueue into a unique runqueue per processor. This design is often called a multiqueue scheduler. Each processor's runqueue has a separate selection of the runnable tasks on a system. When a specific processor executes the scheduler, it selects only from its runqueue. Consequently, the runqueues receive much less contention, and performance does not degrade as the number of processors in the system increases. Figure 1 is an example of a dual-processor machine with a global runqueue vs. a dual-processor machine with per-processor runqueues.

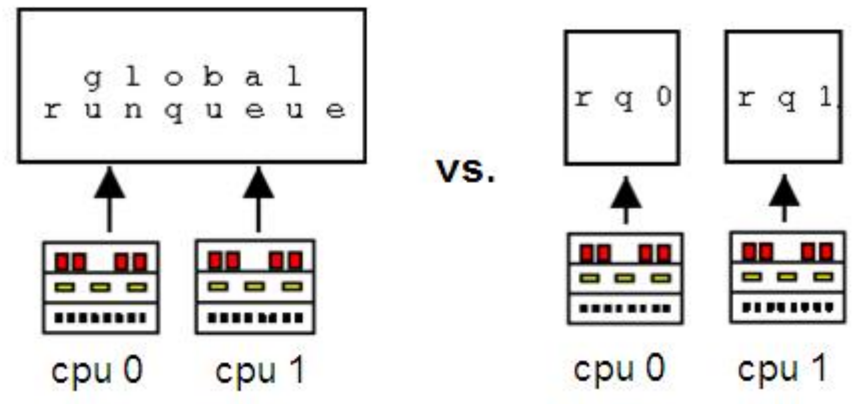


Figure 1. Left, the 2.4 Runqueue; Right, the 2.5/2.6 Runqueue

The third and final goal was improved SMP affinity. The previous Linux scheduler had the undesirable characteristic of bouncing processes between multiple processors. Developers call this behavior the ping-pong effect. Table 1 demonstrates this effect at its worst.

Table 1. A Worst-Case Example of the Ping-Pong Effect

The new scheduler solves this problem, thanks to the new per-processor runqueues. Because each processor has a unique list of runnable processes, processes remain on the same processor. Table 2 shows an example of this improved behavior. Of course, sometimes processes do need to move from one processor to another, like when an imbalance in the number of processes on each processor occurs. In this case, a special load balancer mechanism migrates processes to even out the runqueues. This action occurs relatively infrequently, so SMP affinity is well preserved.

Table 2. The New Scheduler Preserves CPU Affinity

The new scheduler has a lot more features than its name implies. Table 3 is a benchmark showing off the new scheduler.

Table 3. The chatserver benchmark tests message passing between a large number of processes. Results are in messages/second.

Preemptive Kernel

The purpose of kernel preemption is to lower scheduling latency. The result is improved system response and interactive *feel* of the system. The Linux kernel became preemptive with version 2.5.4. Previously, kernel code executed cooperatively. This meant a process—even a real-time one—could not preempt another process executing a system call in the kernel. Consequently, a lower priority process could priority invert a higher priority process by denying it access to the processor when it requested it. Even if the lower priority process' timeslice expired, it would continue running until it completed its work in the kernel or voluntarily relinquished control. If the higher priority process waiting to run is a text editor in which the user is typing or an MP3 player ready to refill its audio buffer, the result is poor interactive performance. Worse, if the higher priority process is a specialized real-time process, the result could be catastrophic.

Why was the kernel not preemptive from the start? Because it is more work to provide a preemptive kernel. If tasks in the kernel can reschedule at any moment, protection must be in place to prevent concurrent access to shared data. Thankfully, the issues a preemptive kernel creates are identical to the concerns raised by symmetrical multiprocessing (SMP). The mechanisms that provide protection under SMP were adapted easily to provide protection with kernel preemption. Thus, the kernel simply leverages SMP spinlocks as preemption markers. When code would hold a lock, preemption is similarly disabled. Otherwise, it is safe to preempt the current task.

Latency Improvements

Most likely, one can now see the next bottleneck. The preemptive kernel simply reduces scheduling latency from the entire length of kernel execution to the duration of spinlocks. It's definitely shorter, sure, but it's still a potential problem. Thankfully, reducing lock duration, which is equal to the length of time kernel preemption is disabled, is doable.

Kernel developers optimized kernel algorithms for lower latency. They primarily concentrated on the VM and virtual filesystem (VFS) and, consequently, greatly reduced the lock duration. The result is excellent system response. Users have observed worst-case scheduling latency in 2.5, even on average machines, at less than 500 nanoseconds.

Redesigned Block Layer

The block layer is the chunk of the kernel responsible for supporting block devices. Traditional UNIX systems support two general types of hardware devices, character devices and block devices. Character devices, such as serial ports and keyboards, manipulate data as a stream of characters, or bytes, one at a time. Conversely, block devices manipulate data in groups of a fixed size (called blocks). Block devices do not merely send or receive a stream of data; instead, any of their blocks are accessible. Moving to one block from another is called seeking. Examples of block devices include hard disks, CD-ROM drives and tape backup devices.

Managing block devices is a nontrivial job. Hard disks are complicated pieces of hardware for which the operating system needs to support arbitrary reading and writing of any valid block. Further, because seeks are expensive, the operating system must intelligently manage and queue requests to block devices to minimize seeks.

The block layer in Linux was in serious need of a redesign. Thankfully, starting with kernel 2.5.1, the revamp began. The most interesting work involved creating a new flexible and generic structure to represent block I/O requests, eliminating bounce buffers and supporting I/O directly into high memory, making the global io_request_lock per queue and building a new I/O scheduler.

Prior to 2.5, the block layer used the buffer_head structure to represent I/O requests. This method was inefficient for several reasons, the largest being the block layer often had to break the data structures into smaller chunks, only to reconstruct them later in the I/O scheduler. In 2.5, the kernel makes use of a new data structure, struct bio, to represent I/O. This structure is simpler, appropriate for both raw and buffered I/O, works with high memory and may be split and merged easily. The block layer consistently uses the new bio structure, resulting in cleaner, more efficient code.

The next issue was eliminating the bounce buffer used when performing I/O into high memory. In the 2.4 kernel, an I/O transfer from a block device into high memory has to make an unfortunate extra stop. High memory is non-permanently mapped memory for which the kernel must provide special support. On Intel x86 machines, this is any memory over about 1GB. Any I/O request into high memory (for example, reading a file from a hard drive into a memory address greater than 1GB) must make use of a special bounce buffer that resides in low memory. The rationale is that some devices may be unable to understand high memory addresses. The result is devices always must perform their I/O transfers into low memory. If the final destination is in fact high memory, the data must bounce from the block device to low memory and finally into high memory (Figure 2). This extra copy introduces significant overhead. The 2.5 kernel now automatically supports transferring directly into high memory, eliminating the bounce buffer logic for devices that are capable.

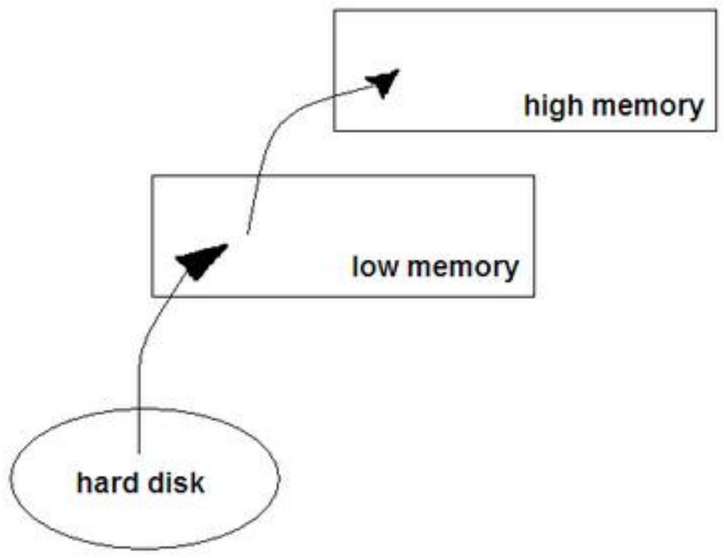


Figure 2. The Bounce Buffer in the 2.4 Kernel

The next bottleneck that developers tackled was the global I/O request lock. Each block device is associated with a request queue, which stores block I/O requests, the individual bio structures that represent each block read or write. The kernel constantly updates the queues as drivers add or remove requests. The io_request_lock protects the queues from concurrent access—code may update a queue only while holding the lock. In kernels prior to 2.5, a single global lock protects all the request queues in the system. The global lock prevents concurrent access to *any* queue, and the lock merely needs to prevent concurrent access to any single queue. In 2.5, a fine-grained lock for each queue replaced the global request lock (Figure 3). Consequently, the kernel now can manipulate multiple queues at the same time.

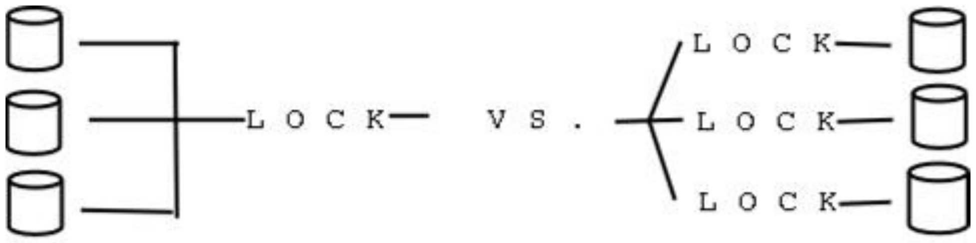


Figure 3. The 2.5 kernel introduces one lock per request queue.

Finally, a new I/O scheduler solved the remaining block layer inefficiency. The I/O scheduler is responsible for merging block requests and sending them to the physical devices. Because seeks are expensive, the I/O scheduler prefers to service contiguous requests. To this end, it sorts incoming requests by sector. This is an important feature for both disk performance and longevity. The problem is, however, that repeated I/O requests to contiguous sectors could prevent servicing of a request for a nonadjacent sector. The new I/O scheduler solves this problem by implementing deadlines for I/O requests. If the I/O scheduler starves a request past its deadline, the I/O scheduler services the starved request rather than continuing to merge requests at the current sector. The new I/O scheduler also solves the writes-starving-reads problem by giving preferential treatment to read requests over write requests. This change greatly improves read latency. Last but not least, the request queue is now a red/black tree, which is an easily searchable data structure, instead of a linear list.

Improved VM Subsystem

During 2.5, VM finally came into its own. The VM subsystem is the component of the kernel responsible for managing the virtual address space of each process. This includes the memory management scheme, the page eviction strategy (what to swap out when memory is low) and the page-in strategy (when to swap things back in). The VM often has been a rough issue for Linux. Good VM performance on a specific workload often implies poor performance elsewhere. A fair, simple, well-tuned VM always seemed unobtainable—until now.

The new VM is the result of three major changes:

- reverse-mapping (rmap) VM
- redesigned, smarter, simpler algorithms
- tighter integration with the VFS layer

The net result is superior performance in the common case without the VM miserably falling to pieces in the corner cases. Let's briefly look at each of these three changes.

Any virtual memory system has both physical addresses (the address of actual pages on your physical RAM chips) and virtual addresses (the logical address presented to the application). Architectures with a memory management unit (MMU) allow convenient lookup of a physical address from a virtual address. This is desirable because programs are accessing virtual addresses constantly, and the hardware needs to convert this to a physical address. Moving in the reverse direction, however, is not so easy. In order to resolve from a physical to a virtual address, the kernel needs to scan each page table entry and look for the desired address, which is time consuming. A reverse-mapping VM provides a reverse map from virtual to physical addresses. Consequently, instead of:

```
for (each page table entry)
        if (this physical address matches)
                we found a corresponding virtual address
```

the rmap VM simply can look up the virtual address by following a pointer. This method is much faster, especially during intensive VM pressure. Figure 4 is a diagram of the reverse mapping.

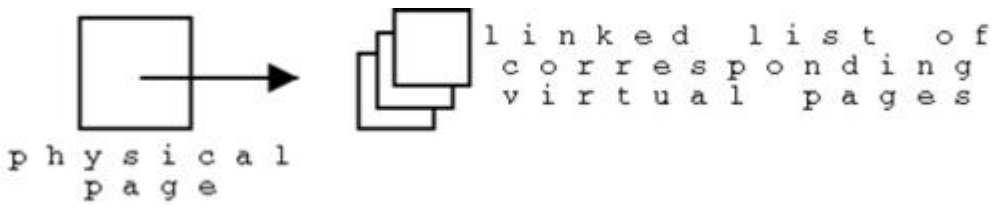


Figure 4. Reverse mapping maps one physical page to one or more virtual pages.

Next, the VM hackers redesigned and improved many of the VM algorithms with simplification, great average-case performance and acceptable corner-case performance in mind. The resulting VM is simplified yet more robust.

Finally, integration between the VM and VFS was greatly improved. This is essential, as the two subsystems are intimately related. File and page write-back, read-ahead and buffer management was simplified. The pdflush pool of kernel threads replaced the bdflush kernel thread. The new threads are capable of providing much-improved disk saturation; one developer noted the code could keep *sixty* disk spindles concurrently saturated.

Threading Improvements

Thread support in Linux always has seemed like an afterthought. A threading model does not fit perfectly into the typical UNIX process model, and consequently, the Linux kernel did little to make threads feel welcome. The user-space pthread library (called LinuxThreads) that is part of glibc (the GNU C library) did not receive much help from the kernel. The result has been less than stellar thread performance. There was a lot of room for improvement, but only if the kernel and glibc hackers worked together.

Rejoice, because they did. The result is greatly improved kernel support for threads and a new user-space pthread library, called Native POSIX Threading Library (NPTL), which replaces LinuxThreads. NPTL, like LinuxThreads, is a 1:1 threading model. This means one kernel thread exists for every user-space thread. That developers achieved excellent performance without resorting to an M:N model (where the number of kernel threads may be dynamically less than the number of user-space threads) is quite impressive.

The combination of the kernel changes and NPTL results in improved performance and standards compliance. Some of the new changes include:

- thread local storage support
- O(1) exit() system call
- improved PID allocator
- clone() system call threading enhancements
- thread-aware code dump support

- threaded signal enhancements
- a new fast user-space locking primitive (called futexes)

The results speak for themselves. On a given machine, with the 2.5 kernel and NPTL, the simultaneous creation and destruction of 100,000 threads takes less than two seconds. On the same machine, without the kernel changes and NPTL, the same test takes approximately 15 minutes.

Table 4 shows the results of a test of thread creation and exit performance between NPTL, NGPT (IBM's M:N pthread library, Next Generation POSIX Threads) and LinuxThreads. This test also creates 100,000 threads but in much smaller parallel increments. If you are not impressed yet, you are one tough sell.

Table 4. Results of the thread creation and exit test: this test measures the time for ten initial threads to each create and destroy one, five or ten parallel threads.

New Sound Layer

The long-awaited merge of the advanced Linux sound architecture (ALSA) began in kernel 2.5.5. ALSA has a number of improvements over open sound system (OSS), the previous sound layer. Most importantly, ALSA provides a much more robust and feature-filled API than OSS. ALSA drivers and the accompanying user-space library (alsa-lib) allow for the creation of advanced audio applications with minimal effort.

ALSA supports a large number of sound devices and provides a backward-compatible OSS interface. For users who still require or prefer OSS, however, drivers most likely will remain through 2.6.

A Look to the Future

It may be a bit irresponsible to begin looking past 2.6 before it is even released. It is interesting, however, to consider what we may see (or at least want to see) in the 2.7 development kernel. With luck, we will see the long-desired tty (terminal) layer rewrite. The tty layer has grown into a large and confusing hack.

Also high on everyone's wish list is a SCSI layer rewrite. Currently, the SCSI layer is too dumb and its drivers are too smart. It also may be possible to unify parts of the IDE and SCSI layers into a generic disk layer. Whatever the case, the SCSI layer needs a bit of cleanup.

After these items, the rest is uncertain. It is risky to make any predictions; the above are mere observations on what we need today. As always, the actual work in 2.7 will depend on the itch the developers feel like scratching.

Regardless of the future, the 2.6 kernel looks great—excellent scalability, swift desktop response, improved fairness and happily cooperating VM and VFS layers.

email: rml@tech9.net

Robert Love is a kernel hacker who works on various projects, including the preemptive kernel and the scheduler. He is a Mathematics and Computer Science student at the University of Florida and a kernel engineer at MontaVista Software. He hates fish.

Archive Index Issue Table of Contents

Advanced search

The Kernel Configuration and Build Process

Greg Kroah-Hartman

Issue #109, May 2003

Starting with the 2.5 series, it's simpler and faster to customize your kernel or add a driver.

The process of building a kernel has two parts: configuring the kernel options and building the source with those options. In versions before the 2.5 kernel, configuration was driven by a Config.in file within every subdirectory and a main help file, Documentation/Configure.help. The language used to describe the build process was based loosely on a shell-style language that would control which configuration options were presented to the user, depending on which options were currently presented.

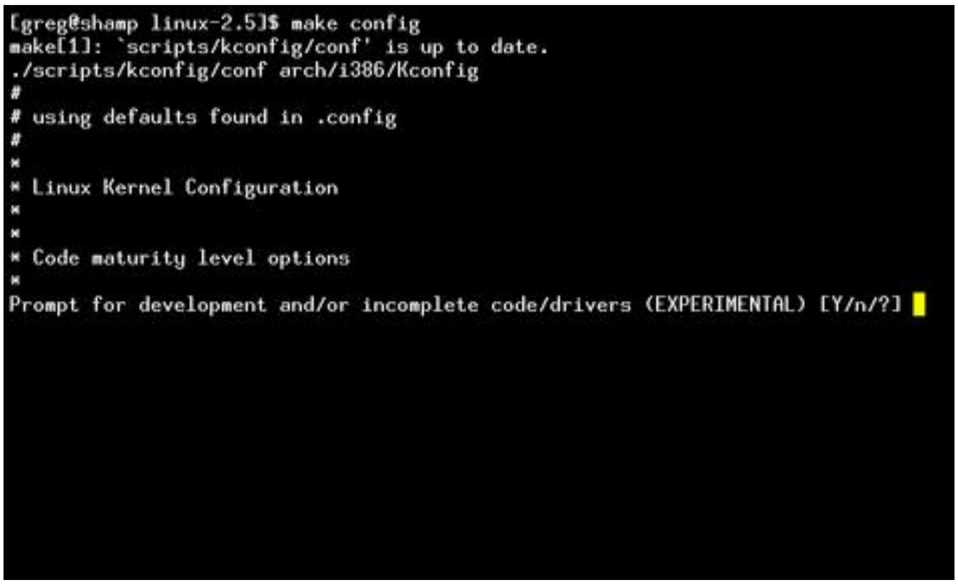
This worked reasonably well, but over time the variety of different options in the kernel stretched the language beyond what it could reasonably handle. In the 2.5.45 kernel release, Roman Zippel's rewrite of the configuration language and configuration programs was placed in the main kernel tree. The new configuration language is much more flexible and powerful. It also unifies the help text with the configuration logic, making it easier to apply patches for individual drivers, without having to worry about conflicts within a single Configuration.help file.

Also during the 2.5 series, Kai Germaschewski and the other kbuild developers slowly reworked makefile logic within the kernel, making it easier to build the kernel based on the selected options. This article describes the format of the makefile and configuration files in the 2.5 kernel and shows how to add a new driver to the build process.

Configuring the Kernel

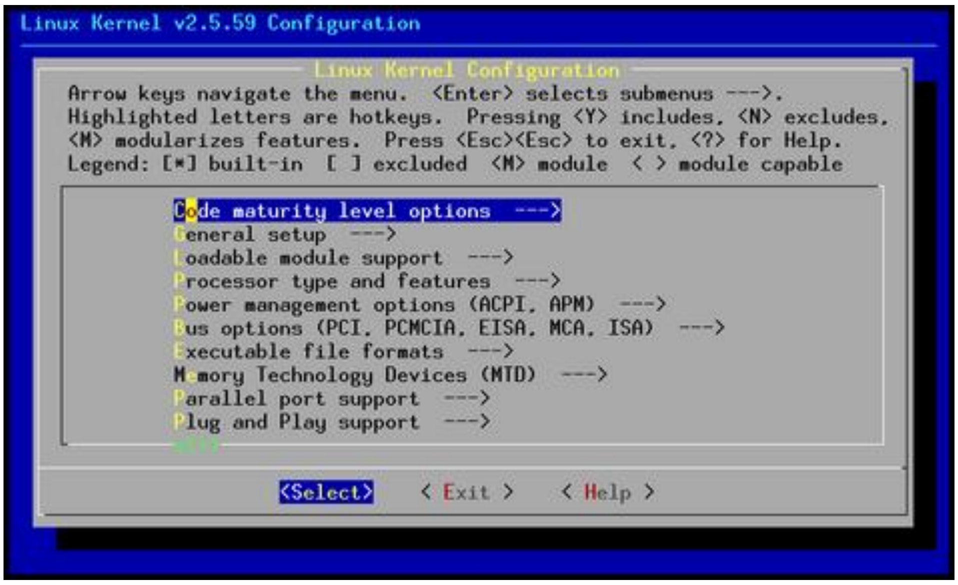
To configure different kernel options, a user runs either a text-mode or a graphical kernel configurator. The text-mode configurator can be run with **make config** and prompts the user to select configuration options in order

(Figure 1). The ncurses text version is more popular and is run with the **make menuconfig** option (Figure 2). The graphical configurator is run with **make xconfig** and uses Qt as the widget set (Figure 3).



```
[greg@shamp linux-2.5]$ make config
make[1]: `scripts/kconfig/conf' is up to date.
./scripts/kconfig/conf arch/i386/Kconfig
#
# using defaults found in .config
#
*
* Linux Kernel Configuration
*
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers (EXPERIMENTAL) [Y/n/?]
```

Figure 1. Configuring the Kernel with **make config**

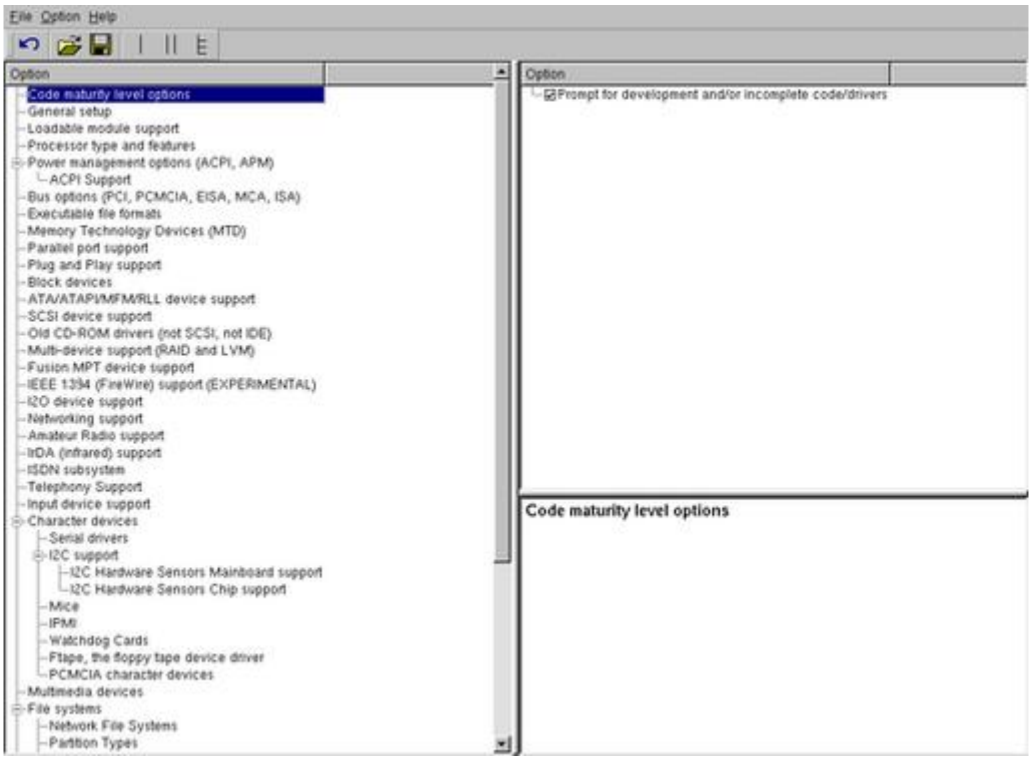


```
Linux Kernel v2.5.59 Configuration
                    Linux Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

           Code maturity level options  --->
           General setup  --->
           Loadable module support  --->
           Processor type and features  --->
           Power management options (ACPI, APM)  --->
           Bus options (PCI, PCMCIA, EISA, MCA, ISA)  --->
           Executable file formats  --->
           Memory Technology Devices (MTD)  --->
           Parallel port support  --->
           Plug and Play support  --->

                  <Select>    < Exit >    < Help >
```

Figure 2. **make menuconfig** makes it easier to back up and correct mistakes.


File Option Help
Option
Code maturity level options
General setup
Loadable module support
Processor type and features
Power management options (ACPI, APM)
ACPI Support
Bus options (PCI, PCMCIA, EISA, MCA, ISA)
Executable file formats
Memory Technology Devices (MTD)
Parallel port support
Plug and Play support
Block devices
ATA/ATAPI/MFM/RLL device support
SCSI device support
Old CD-ROM drivers (not SCSI, not IDE)
Multi-device support (RAID and LVM)
Fusion MPT device support
IEEE 1394 (FireWire) support (EXPERIMENTAL)
I2O device support
Networking support
Amateur Radio support
IrDA (infrared) support
ISDN subsystem
Telephony Support
Input device support
Character devices
Serial drivers
I2C support
I2C Hardware Sensors Mainboard support
I2C Hardware Sensors Chip support
Mice
IPMI
Watchdog Cards
Ftape, the floppy tape device driver
PCMCIA character devices
Multimedia devices
File systems
Network File Systems
Partition Types
Option
Code maturity level options

☑ Prompt for development and/or incomplete code/drivers

Figure 3. The Qt-Based **make xconfig**

When the kernel configurator is run, it reads the main kernel configuration file, located in arch/i386/Kconfig for the i386 platform. Other architectures have the main configuration files located in their main directories. This main configuration file then includes other configuration files from the different subdirectories in the kernel directory tree. Those configuration files also can include other configuration files as needed. For example, the arch/i386/Kconfig file contains the line:

```
source "sound/Kconfig"
```

which will read information from that file. This sound/Kconfig file then includes a lot of other files:

```
source "sound/core/Kconfig"
source "sound/drivers/Kconfig"
source "sound/isa/Kconfig"
source "sound/pci/Kconfig"
source "sound/ppc/Kconfig"
source "sound/arm/Kconfig"
source "sound/usb/Kconfig"
```

The sound/usb/Kconfig file describes all of the ALSA USB driver options, like this:

```
# ALSA USB drivers
menu "ALSA USB devices"
    depends on SND!=n && USB!=n

config SND_USB_AUDIO
    tristate "USB Audio/MIDI driver"
    depends on SND && USB
    help
      Say 'Y' or 'M' to include support for
```

```
        USB audio and USB MIDI devices.
endmenu
```

The # character can be used to comment Kconfig files. Anything written after it on the same line is not read by the configurator, but it is useful for documenting what the file is for and what it should do.

The menu and endmenu commands tell the configurator to declare a new menu level or new screen in some of the configuration programs. On the menu line, the name of the menu should be specified within " characters. For this file, the menu is called **"ALSA USB devices"**.

Menus and configuration options can be controlled to display or not. In this example, the USB option menu is only displayed if the CONFIG_SND and CONFIG_USB options are selected, which is controlled by the line **depends on SND!=n && USB!=n**. To help decrease the amount of typing involved, all configuration options automatically start with CONFIG, which is not used within the configuration language. The valid states for a configuration option are:

- y—the option is enabled.
- n—the option is not enabled.
- m—the option is set to be built as a module.

If both the CONFIG_SND and CONFIG_USB options are not set to n (meaning they are set either to be built in to the kernel or to build as a module), the CONFIG_SND_USB_AUDIO option is presented to the user. This option can be set to one of the three values, and it is described as a "tristate" value. The text that should be shown to the user is **"USB Audio/MIDI driver"**:

```
tristate "USB Audio/MIDI driver"
```

The valid values for describing a configuration variable are:

- bool—the variable can be set only to y or n.
- tristate—the variable can be set to y, n or m.
- int—the variable can be set to any numeric value.

This configuration option is controlled by a depends logic line, which follows the same logic as a menu option. The CONFIG_SND_USB_AUDIO option depends on both the CONFIG_SND and CONFIG_USB options, meaning that if one of these options is set to a module, then the CONFIG_SND_USB_AUDIO option also should be set to a module. If both of the controlling options are not enabled (meaning both are set to n), this option will not be displayed. If both of these options are set to y, this option can be selected as n, y or m. All of this is defined with the simple line:

```
  depends on SND && USB
```

Within the kernel code, the configuration variable will be seen (the CONFIG_SND_USB_AUDIO in the above example), so the code can test for it or any other kernel configuration option's existence. However, using **#ifdef** within a .c file to test for different configuration options is against the kernel-style programming guidelines, which I covered in my article "Proper Linux Kernel Coding Style" [*LJ*, July 2002, www.linuxjournal.com/article/5780]. Instead, limit the use of #ifdef to .h files, keeping the .c files cleaner and easier to read.

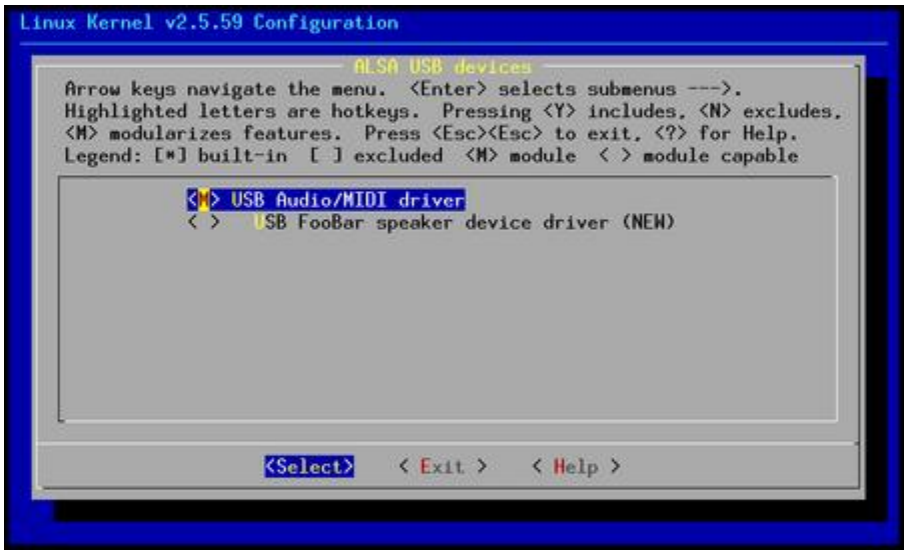
Previously, the help text for a configuration option was placed in one big Configuration.help file. Now the help text is placed right after the depends line within the Kconfig file. It begins with a line containing either **help** or **---help---**, followed by a number of lines of help text that are indented two spaces from the help line.

Adding a New Configuration Option

To add a new configuration option, simply add new lines to an existing Kconfig file, in the same location as a related configuration option. For example, if a new USB sound device driver is written for the ALSA sound system, it would go into the sound/usb directory, and the sound/usb/Kconfig file would be added. This new device driver controls the mythical FooBar USB speaker device. It depends on having the CONFIG_SND and CONFIG_USB options enabled in addition to the CONFIG_SND_USB_AUDIO option, as the new driver uses some functions found in that driver. The new configuration option should be placed after the SND_USB_AUDIO option but before the closing endmenu command, and it would look something like:

```
  config SND_USB_FOOBAR
      tristate "USB FooBar speaker device driver"
      depends SND_USB_AUDIO
      help
          Say Y here if you want to use FooBar USB
          speaker device.
          This code is also available as a module
          (= code which can be inserted in and
          removed from the running kernel whenever
          you want). The module will be called
          usbfoobar.o.
```

This option will now show up when the SND_USB_AUDIO option is selected (Figure 4).



```
Linux Kernel v2.5.59 Configuration
                    ALSA USB devices
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.
Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

          <M> USB Audio/MIDI driver
          < >   USB FooBar speaker device driver (NEW)

          <Select>    < Exit >    < Help >
```

Figure 4. The Newly Enabled FooBar USB Speaker Device

Building the Kernel

The kernel is built with a system of individual makefiles that are all linked together when the kernel is built, forming a large makefile. The individual makefiles do not look like any standard makefile, but instead follow a special format that is unique to the kernel build process. The makefile needs to build only the necessary files, depending on the configuration options enabled, in the proper format (as modules or built in to the kernel). As an example, drivers/usb/misc/Makefile in the 2.5.59 kernel release looks like:

```
#
# Makefile for the rest of the USB drivers
# (the ones that don't fit into any other
# categories)
#
obj-$(CONFIG_USB_AUERSWALD)  += auerswald.o
obj-$(CONFIG_USB_BRLVGER)    += brlvger.o
obj-$(CONFIG_USB_EMI26)      += emi26.o
obj-$(CONFIG_USB_LCD)        += usblcd.o
obj-$(CONFIG_USB_RIO500)     += rio500.o
obj-$(CONFIG_USB_SPEEDTOUCH) += speedtch.o
obj-$(CONFIG_USB_TEST)       += usbtest.o
obj-$(CONFIG_USB_TIGL)       += tiglusb.o
obj-$(CONFIG_USB_USS720)     += uss720.o
speedtch-objs := speedtouch.o atmsar.o
```

The line:

```
obj-$(CONFIG_USB_LCD)        += usblcd.o
```

builds the usblcd.c file into a module if the CONFIG_USB_LCD configuration option is set to m. Otherwise, it is built into the kernel directly if that configuration option is set to y. This step is all that is necessary to add to a kernel makefile if the module is made from only a single .c file.

If the driver consists of multiple .c files, the name of the files needs to be listed on separate lines, along with the name of the module that this driver is called. In the previous example file, this listing of file and driver names looks like:

```
obj-$(CONFIG_USB_SPEEDTOUCH) += speedtch.o
```

and

```
speedtch-objs := speedtouch.o atmsar.o
```

The first line controls whether the speedtch module is built. If it is, the line indicates whether it is compiled into the kernel or stands as a module. The second line explains that the speedtouch.c and atmsar.c files will be built into .o files and then linked together into the speedtch.o module.

In older kernels, if a file exported symbols, it needed to be explicitly mentioned in the kernel makefiles. In 2.5 and later kernels, that mention is no longer necessary.

Adding a New Driver to the Build Process

To add a new driver to the kernel build process, a single line needs to be added if the driver is contained within a single file. Based on the previous example of the FooBar USB speaker device, the line:

```
obj-$(CONFIG_SND_USB_FOOBAR) += usbfoobar.o
```

is added to sound/usb/Makefile.

If the driver is contained in two files, such as foobar1.c and foobar2.c, an additional line needs to be added:

```
usbfoobar-objs := foobar1.o foobar2.o
```

Conclusion

The kernel configuration and build process in the 2.5 kernel is much simpler and more flexible than in the previous kernel versions. Thanks go to Roman Zippel and Kai Germaschewski for doing the work to make it easier for kernel developers to focus on writing code and not have to worry about the intricacies of the kernel build process.

A good resource for more information on the specifics of the Kbuild process is available from Sam Ravnborg, at marc.theaimsgroup.com/?l=linux-kernel&m=104162417329638.

Greg Kroah-Hartman is currently the Linux USB and PCI Hot Plug kernel maintainer. He works for IBM, doing various Linux kernel-related things and can be reached at greg@kroah.com.

Archive Index Issue Table of Contents

Advanced search

Reiser4, Part II: Designing Trees that Cache Well

Hans Reiser

Issue #109, May 2003

Version 4 of the Reiser filesystem flattens the tree data structure fo rbetter performance than version 3. Here's how it's structured.

This article is the second in a series on the design of the Reiser4 filesystem. The first article [*LJ*, December 2002] defined basic concepts: trees, nodes and items. This article explains why balanced trees are better than unbalanced trees and why B+trees are better than B-trees by explaining and applying the principles of caching. The article then applies these same principles to a classic database technique used in ReiserFS v3 called binary large objects (BLOBs). It suggests that BLOBs reduce the effectiveness of caching internal nodes by making the tree no longer truly balanced. It also shows how Reiser4 stores objects larger than a node without unbalancing the tree.

I apologize to readers for the delay of this article, which is due to the Halloween feature-freeze for 2.6 and the need to stabilize Reiser4 quickly at that time.

Fanout

The fanout rate (n) refers to the number of nodes pointed to by each level's nodes (Figure 1). If each node can point to n nodes of the level below it, then starting from the top, the root node points to n internal nodes at the next level, each of which points to n more internal nodes at its next level and so on. m levels of internal nodes can point to nm leaf nodes containing items in the last level. The more you want to store in the tree, the larger you have to make the fields in the key that first distinguish the objects, then select parts of the object (the offsets). This means your keys must be larger, which decreases fanout (unless you compress your keys, but that will wait for our next version).

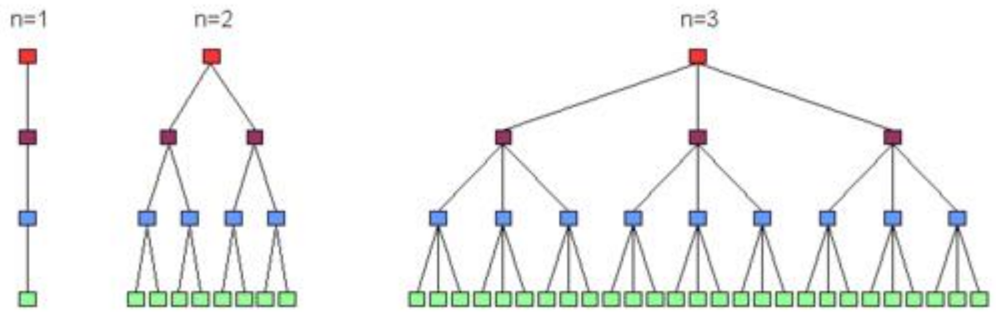


Figure 1. Three Four-Level, Height-Balanced Trees

In Figure 1, the first graph is a four-level tree with a fanout of n = 1. It has only four nodes, starting with the (red) root node, traversing the (burgundy) internal and (blue) twig nodes and ending with the (green) leaf node, which contains the data. The second tree, with four levels and a fanout of n = 2, starts with a root node, traverses two internal nodes, each of which points to two twig nodes (for a total of four twig nodes) and each of these points to two leaf nodes for a total of eight leaf nodes. Lastly, a four-level, fanout of n = 3 tree is shown, which has one root node, three internal nodes, nine twig nodes and 27 leaf nodes.

B+Trees Are Better than B-Trees

You can store not only pointers and keys in internal nodes but also the objects to which those keys correspond. This is what the original B-tree algorithms did (Figure 2).

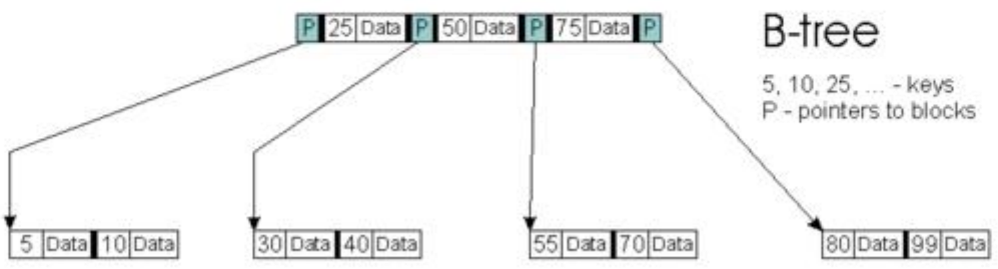


Figure 2. A B-Tree

Then, B+trees were invented that have only pointers and keys stored in internal nodes with all of the objects stored at the leaf level (Figure 3).

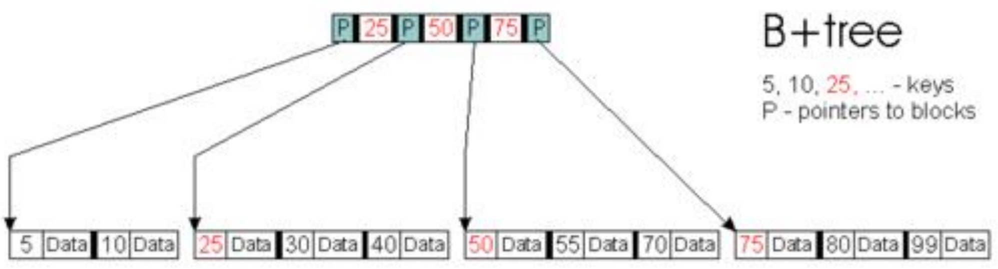


Figure 3. A B+Tree

B+Trees Have a Higher Fanout than B-Trees

Fanout is increased when we put only pointers and keys in internal nodes and don't dilute them with object data. Increased fanout raises our ability to cache all of the internal nodes, because there are fewer internal nodes. People often respond to this by saying, “but B-trees cache objects, and caching objects is just as valuable.” This is not, on average, the answer. Of course, discussing averages makes the discussion much more difficult.

However, we need to cover some cache design principles before getting to this. Let's suppose the following:

- You have two sets of things, A and B.
- You need things from those two sets semi-randomly, with a tendency for some things to be needed much more frequently than others, but which things those are can shift over time.
- You can keep things around after you use them in a cache of limited size.
- You tie the caching of each thing from A to the caching of some particular thing from B. This means that whenever you fetch something from A into the cache, you fetch its partner from B into the cache.

This increases the amount of cache required to store everything recently accessed from A. If there is a strong correlation between the need for the two particular objects that are tied in each of the pairings, stronger than the gain from spending those cache resources on caching more members of A and B according to the LRU (least recently used) algorithm, then this might be worthwhile. If no such strong correlation exists, it is bad. LRU means that we choose the least recently used thing to discard from the cache when we need to make more room. Various approximations of LRU are the most commonly used caching algorithms in OS design.

But wait, you might say, you need things from B also, so it is good that some of them were cached. Yes, you need some random subset of B. The problem is that without a correlation, the things from B that you need are not especially likely to be those same things from B that were tied to the things from A that were needed. Choosing what from B you bring into the cache and keep in the cache on the basis of something other than LRU may reduce the effectiveness of caching, unless it is done according to an algorithm at least as good as LRU. Often choosing which members of B to cache based on which members of A have been cached is not as good as LRU, and so we have a problem.

This tendency to inefficiently tie things that are randomly needed exists outside the computer industry. For instance, suppose you like both popcorn and sushi, with your need for them on a particular day being random. Suppose that you like movies randomly. Suppose a theater requires you to eat only popcorn while watching the movie you randomly found optimal to watch, and not eat sushi from the restaurant on the corner while watching that movie. Is this a socially optimum system? Suppose quality is randomly distributed across all hot dog vendors. If you can only eat the hot dog produced by the best movie displayer on a particular night that you want to watch a movie, and you aren't allowed to bring in hot dogs from outside the movie theater, is this a socially optimum system? Optimal for you?

Tying strongly correlated things together can sometimes be good for performance, however. Many filesystems tie access to information about the file's size to information about the file's name. This seems to work well, better than LRU would.

Tying uncorrelated things together is a common error in designing caches but is still not enough to describe why B+trees are better. With internal nodes, we store more than one pointer per node, meaning pointers are not cached separately. You could argue that pointers and the objects to which they point are more strongly correlated than the different pointers. I hope what we have discussed here is instructive, but we still need another cache design principle.

My Definition of Cache Temperature

Let the cache temperature of something be equal to how often you access it, times the average cost to fetch it from disk, divided by the number of bytes of cache it consumes. You may notice a certain careful lack of precise detail in this definition—particularly, how small objects read individually tend to be hotter because of the cost of performing seeks. Other definitions of cache temperature are possible, but this one is most convenient for this article.

If two types of things cached in nodes have different average temperatures, segregating them into separate nodes helps caching. Suppose you have R bytes of RAM for cache and D bytes of disk. Suppose that 80% of accesses are to the most recently used things that are stored in H (hotset) bytes of nodes. Reducing the size of H so it is smaller than R is important for performance. If you disperse your frequently accessed data evenly, a larger cache is required, and caching is less effective.

Caching Principles

If, all else being equal, we increase the variation in temperature among all nodes, we increase the effectiveness of using a fast small cache.

If two types of things have different average temperatures, separating them into separate nodes increases the variation in temperature in the system as a whole.

If all else is equal, and if two types of things cached several to a node have different average temperatures, segregating them into separate nodes instead helps caching.

Pointers to Nodes

Pointers to nodes frequently tend to be accessed relative to the number of bytes required to cache them. Consider that you have to use the pointers for all tree traversals that reach the nodes beneath them, and that they are smaller than the nodes to which they point.

Putting only node pointers and delimiting keys into internal nodes concentrates the pointers. Because pointers tend to be two orders of magnitude more frequently accessed per byte of their size than items storing file bodies, a high average temperature difference exists between pointers and object data.

According to the caching principles described previously, segregating these two types of things with different average temperatures (pointers and object data) increases the efficiency of caching.

Now you might say, why not segregate by actual temperature instead of by type, because type correlates only with temperature? We do what we can easily and effectively code, with not only temperature segregation in consideration. Some tree designs rearrange the tree so that objects with a higher temperature are higher in the tree than pointers with a lower temperature. The difference in average temperature between object data and pointers to nodes is so high that I don't find such designs a compelling optimization, plus they add complexity. Given the two order of magnitude average temperature difference, I suspect that if I am wrong it is not by enough to care about.

On a side note, although these other tree designs just mentioned migrate objects higher in the tree according to temperature, if one was merely to segregate by temperature without changing levels, it might be more effective. If one had no compelling semantic basis for aggregating objects near each other (this is true for some applications), and if one wanted to access objects by nodes rather than individually, it would be interesting to have a node repacker sort object data into nodes by temperature.

B+Trees Cache Better than B-Trees

B+trees separate the pointers and the data into different nodes. On average, the pointers to the nodes of the tree are hotter than the data of the things stored in the tree (by about two orders of magnitude). Therefore, according to the principles of caching explained previously, caching is improved by separating the pointers to nodes from the data of things stored in the tree.

In the industry, B+trees are known in practice to be better than B-trees, exactly as this theory predicts. It is also accepted wisdom that balanced trees perform better than unbalanced trees.

What is not currently accepted wisdom, but is predicted by the application of these principles, is that the use of, what are called by the database industry, BLOBs hurts performance. More on that (and what BLOBs are) in a bit.

What Does Balanced Mean?

The term balanced is used for several distinct purposes in balanced tree literature. Two of the most common are balanced height and balanced space usage within the nodes of the tree. Unfortunately, these different definitions are a classic source of confusion for readers of the literature, and I'll try to avoid that in this article.

Height-balanced trees are those for which each possible search path from root node to leaf node has exactly the same length; length equals the number of nodes traversed from root node to leaf node. For instance, the height of the tree in Figure 1 is four, the height of the tree in Figure 4 is three and the height of the single-node tree is one.

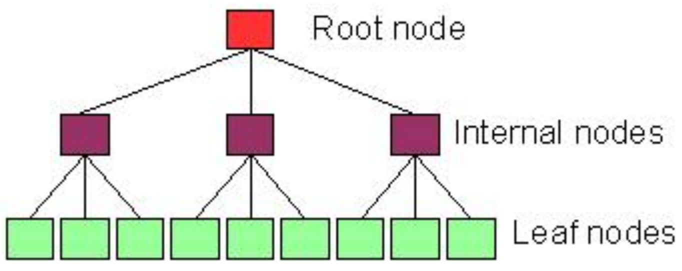


Figure 4. A Three-Level Tree

Most algorithms for accomplishing height balancing do so by growing the tree only at the top. Thus, the tree never gets out of height balance.

Figure 5 shows an unbalanced tree. It originally could have been balanced and then lost some of its internal nodes due to deletion, or it could have been balanced once but now be growing by insertion, without yet undergoing rebalancing.

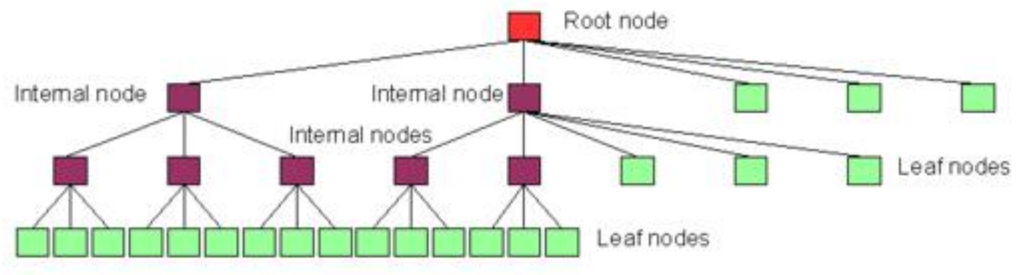


Figure 5. An Unbalanced Tree

Traditional database methods for storing objects larger than nodes (BLOBs) make trees unbalanced. BLOBs are a method of storing objects larger than a node by storing pointers to nodes containing the object. These pointers commonly are stored in what are called the leaf nodes (level 1, except that the BLOBs are then sort of a basement “level B”) of a “B*” tree.

In Figure 6, a BLOB has been inserted into a leaf node of a four-level tree, meaning pointers to blocks containing the file data have been inserted into the leaf node. This is what a ReiserFS v3 tree looks like.

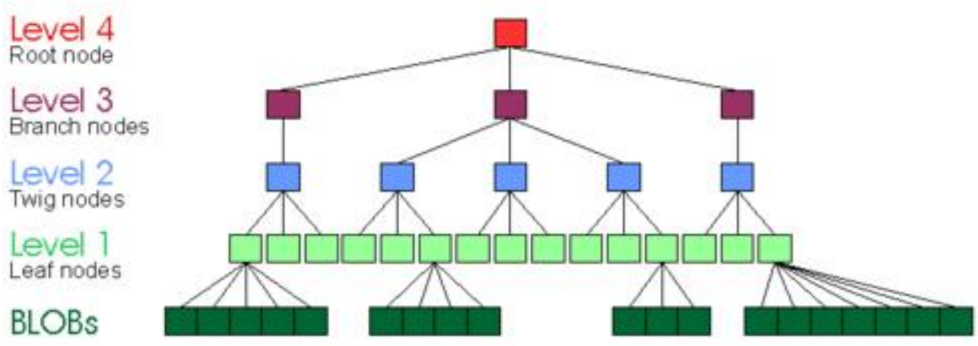


Figure 6. A Four-Level Tree after Insertion of a BLOB

This is a significant definitional drift, albeit one accepted by the entire database community. By the principles of caching described here, this reduces the separation of pointers and object data, which in turn reduces the effectiveness of caching. I suggest that those principles of caching indicate it is a bad design. For all of the reasons that B+trees are better than B-trees, Reiser4 trees are better than ReiserFS v3 trees, though to a less extreme amount.

By contrast, Figure 7 is a Reiser4 tree with a fanout of three, a BLOB in the level-one leaf nodes and the pointer to it in the level-three twig nodes. In this case, the BLOB's blocks are all contiguous. For reasons of space, it is set below the other leaf nodes, but its extent pointer is in a level-two twig node, like every other item's pointer.

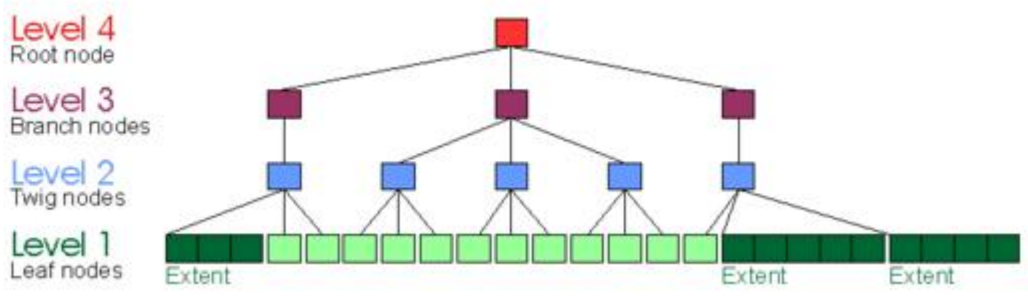


Figure 7. A Reiser4 tree stores the BLOB in the level-one leaf nodes.

Although it is accepted that B+trees are better than B-trees, it is not well accepted that BLOBs are a bad design, and indeed, it is the dominant paradigm within the database industry.

Gray and Reuter say the criterion for searching external memory is to “minimize the number of different pages along the average (or longest) search path....by reducing the number of different pages for an arbitrary search path, the probability of having to read a block from disk is reduced” (see Resources).

My problem with this explanation of the effectiveness of the height-balanced approach is it does not convey that you can get away with having a moderately unbalanced tree, provided you do not significantly increase the total number of internal nodes. In practice, most unbalanced trees do have significantly more internal nodes. In practice, most moderately unbalanced trees have a moderate increase in the cost of in-memory tree-traversals and an immoderate increase in the amount of I/O due to the increased number of internal nodes not remaining in cache because there are now too many of them.

But, if one were to put all the BLOBs together in the same location in the tree, because the amount of internal nodes would not significantly increase, the performance penalty for having them on a lower level of the tree than all other leaf nodes would not be a significant additional I/O cost. There would be a moderate increase in that part of the tree traversal time cost, which is dependent on RAM speed, but this would not be so critical. Segregating BLOBs could perhaps substantially recover the performance lost by architects not noticing the drift in the definition of height balancing for trees. There might also be a substantial I/O-related performance effect for segregating BLOBs that is unrelated to tree considerations. Perhaps someday someone will try it and tell us what happens.

Reiser4 returns to the classical definition of a height-balanced tree in which the lengths of the paths to all leaf nodes are equal. It does not pretend that all of the nodes storing objects larger than a node are somehow not part of the tree, even though the tree stores pointers to them.

Reiser4 reduces the number of internal nodes, nodes containing pointers, from the number required for ReiserFS v3. The number of internal nodes required for ReiserFS v3 to store the 188MB Linux kernel 2.4.1 source code tree is 1,629. Reiser4 requires only 164. As a result, the amount of RAM required to store pointers to nodes is reduced dramatically in Reiser4.

A Hint about Articles Ahead

In upcoming articles we will discuss why, even when nothing is cached in memory, Reiser4's performance is much higher than that of ReiserFS v3, why dancing trees are better than space-usage balanced trees and how we added support for transactions while at the same time greatly reducing the amount of data that is written twice.

Hans Reiser (reiser@namesys.com) entered UC Berkeley in 1979 after completing the eighth grade and majored in “Systematizing”, an individual major based on the study of how theoretical models are developed. His senior thesis discussed how the philosophy of the hard sciences differs from that of computer science, with the development of a naming system as a case study. He is still implementing that naming system, of which Reiser4 is the storage layer.

The Linux Softsynth Roundup

Dave Phillips

Issue #109, May 2003

Are you ready to rock? Now that you've got ALSA and kernel preemption, add software to turn your Linux box into a synthesizer studio.

Software sound synthesis (SWSS) has an honorable lineage in the history of computers. Early experiments in digital sound synthesis took place at the famous Bell Labs, where a team of researchers centered around Max Mathews created the Music N series of audio synthesis software, culminating in Music V in 1969. Since that time, Music V has evolved into a series of notable digital sound synthesis environments, such as Csound, Cmix/RTCmix and Common LISP Music. These environments typically provide the user with a language for specifying the nature of sonic events, such as musical notes or sampled sounds. These languages usually present users with a distinction between instruments (the sound-producing designs) and scores (event characteristics, such as, start time, duration and synthesis parameters). Users compose their instruments and scores in their preferred SWSS language and then feed them to the language's compiler. Output is directed to a file, which then can be played by any sound system supporting the file format or, with sufficiently powerful hardware, the output can be directed to a digital-to-audio converter for rendering real-time audio output.

A standalone software synthesizer (softsynth) substitutes real-time control for the score aspect of the model above. Softsynths typically come with attractive GUIs, often emulating the appearance and operation of a hardware synthesizer, and a MIDI keyboard or external sequencer is the expected controller. Under the right circumstances, a softsynth can be controlled by a concurrent process. For example, using the ALSA aconnect utility, a softsynth can be wired to a MIDI sequencer running on the same machine. Then, sequences can be recorded and played via the softsynth, eliminating the need for an external synthesizer and containing the MIDI environment on a single computer.

A softsynth can be dedicated to a particular synthesis method (additive, subtractive, FM, etc.), or it can be open-ended and modular. In short, additive synthesis works by summing sine waves with varying frequencies, amplitudes and phases until the desired sound is attained. Additive synthesis is a computationally expensive synthesis method with a formidable amount of detail required for realistic sounds. Subtractive synthesis begins with a sound source rich in frequencies (such as a sawtooth wave or noise), then filters frequencies out until the desired sound has been sculpted from the original source. Subtractive synthesis is relatively easy to implement in hardware and software, and its sounds are characteristically associated with the analog synthesizers of the 1970s. FM (frequency modulation) synthesis works by shaping the frequency components of one oscillator by the output of another, creating complex audio spectra with little computational expense. Yamaha's DX7 synthesizer is the most famous FM implementation, and the company's OPL3 sound chip is certainly the most infamous.

Physical modelling and granular synthesis are two more recent synthesis methods. Physical modelling synthesis models the mechanics of a real or imaginary instrument and the physics of its activation. The method's parameters are based less on familiar sono-musical models, such as waveforms, frequencies and amplitudes, and more on the characteristics of physically excited systems, such as airflow through a tube, the vibrations of a plucked string or the radiating patterns of a struck membrane. Physical modelling has become a popular synthesis method and is deployed in synthesizers from Korg, Yamaha and others. Granular synthesis creates sounds by ordering sonic quanta or grains into more or less dense sonic masses. Again, its parameters are not so intuitive as in the older synthesis methods, but it is powerful and can create a wide range of sounds. Granular synthesis has yet to find its way into a popular commercial synthesizer, but hardware implementations are found in the Kyma system and the UPIC workstation.

Synthesizer Architectures

A softsynth can be dedicated wholly to a single synthesis method, it can be a hybrid of two or more methods, or it can take a more open-ended modular design. Each architecture has its strengths. Broadly speaking, the modular design is perhaps the most flexible, but it may sacrifice fineness of control (resolution) for generality of purpose. A dedicated-method softsynth lacks the modular synth's flexibility but usually provides much finer parameter control.

Modular synthesizers encourage a building-block approach by providing separate synthesis primitives for connection in arbitrary ways. For example, an oscillator's output can be directed to the input of an envelope generator (EG) or vice versa, routing the EG's output to an oscillator input. This kind of black box networking lends itself to software emulation, as we'll see when we meet some modular synths later in this article.

The distinctions between the general types of software are blurring. For example, Csound is now available with a set of FLTK-based widgets for user-designed control panels. Many users already have created elaborate GUIs for Csound's various synthesis methods, some of which are detailed enough to function as standalone Csound-based softsynths. This trend is likely to continue with GUIs evolving for the Common LISP Music and RTCmix SWSS environments.

Graphic patching SWSS environments like jMax and Pd are another indicator of this blurring tendency. They also provide graphics widgets that can be used to construct synthesizer interfaces, but unlike Csound, these widgets are an integral aspect of the basic working environment. jMax and Pd utilize a unique combination of graphics and language primitives that are patched together by virtual wires to create a synthesis or processing network. These environments certainly can be employed as softsynths, but their generality of purpose places them closer to Csound than to the softsynths reviewed here.

Beatbox-style synths are yet another softsynth design category. These programs combine elements of a synthesizer, a drum machine and a sequencer for an all-in-one accompaniment package, though the more sophisticated examples are truly more flexible music composition systems.

These distinctions are brief, but for this article they suffice to indicate the basic types of softsynths. For complete definitions of the various synthesis methods and synthesizer architectures, see the standard references listed in Resources.

Plugins

If you've ever used Adobe Photoshop or The GIMP, you're already familiar with the concept of plugins. For normal users, a plugin architecture extends a program's capabilities without requiring an update or a recompile. For applications programmers, a plugin architecture allows them to concentrate on the basic design of their programs, letting the plugins provide more extended or advanced features.

Musicians working with Windows/Mac audio software can use plugins written to the Steinberg VST and Microsoft DirectX plugin APIs. Linux does not directly support either of those APIs, although we shall see an indirect method of support that does work under WINE. However, Linux audio developers have come up with their own native plugin architecture called the Linux Audio Developers Simple Plugin Architecture (LADSPA). The LADSPA API has become a standard, and support for it is now an expected aspect of almost any new Linux audio application. Some outstanding collections of LADSPA plugins are available that include not only the typically expected effects and DSP but also synthesis building blocks (oscillators, envelope generators, filters, etc.) and even some fully formed plugin synthesizers. There are some notable non-LADSPA plugins too.

Peter Hanappe's iiwusynth is a lightweight synthesizer that uses SoundFonts as fuel for its synthesis engine. Given a decent set of SoundFonts, iiwusynth's output is very good, and it has become popular as an embedded synth in many applications. It also can be used as a standalone synthesizer from the command line.

RX/Saturno is another lightweight plugin synthesizer that emulates the popular Yamaha DX7 FM synthesizer. Developer Juan Linietsky has indicated that RX/Saturno is still in the initial development stage, but it already is quite useful and can be employed as a plugin synth in any program supporting the ALSA sequencer.

Kjetil Matheussen's vstserver is interesting software that uses the capabilities of WINE to fool VST plugins into believing they're working in their native Windows environment. In most cases, performance is excellent, at least as good as under Windows. Kjetil also has written two clients for the server, one to hook VST plugins into Pd and one for LADSPA. The vstserver also supports some VSTi plugins, which are fully formed instruments such as synthesizers, samplers and MIDI sequencers wrapped into the VST plugin architecture.

Although LADSPA is an effective and popular standard, the “simple” aspect of its design prohibits certain kinds of processing and control. LADSPA plugins themselves do not permit direct parameter control via MIDI; though the plugins are quite usable in a MIDI sequencer such as MusE. Once again, the Linux audio development community has risen to the challenge with a new proposed standard called XAP. The API is in the design stages, but the team working on XAP includes the LADSPA designers and other talented Linux audio programmers.

Get ALSA

The MIDI input hardware is typically a MIDI synthesizer keyboard, but any MIDI instrument can be used. Connection to a standard sound card requires a MIDI interface cable. OSS/Free and ALSA support MPU-401-compatible devices, so some standalone MIDI cards will work also. ALSA provides direct support for serial port and USB MIDI devices (I have not tested those connections myself) as well as the very useful virmidi virtual MIDI ports.

On the software side, the basic OSS/Free Linux sound system (the kernel sound system) is sufficient for working with the softsynths described here, but the recommended system includes the ALSA library and drivers, the JACK audio connection kit and a hardware MIDI input device. For best response, the kernel should be compiled for low latency, optionally with the preemptive kernel patch. The real-time clock (RTC) should be enabled also.

As of the 2.5 kernel development track, the OSS/Free sound system has been officially replaced by ALSA. Stable kernels from 2.6 onward will build only the ALSA system, which does have an excellent OSS/Free emulation layer for compatibility with non-ALSA-aware applications. Kernels earlier than 2.5 include the OSS/Free system, so users of those kernels must build and install ALSA themselves. ALSA has been designed for the kind of interconnectivity common to a modern sound system. ALSA provides API support for plugins, an advanced audio client/server architecture and a set of utilities to ease system configuration and management.

4Front Technology's proprietary OSS/Linux also works well with Linux softsynths, though obviously it can't take direct advantage of a network of ALSA sequencer clients.

Knowing JACK

JACK is a recursive acronym for the JACK Audio Connection Kit. It has been designed for low latency professional-grade performance as a software bus for the connection of out-of-process audio applications. JACK is somewhat similar in purpose to sound servers such as the aRts server for KDE or GNOME's esd, but JACK has been designed as a more robust solution incorporating pro-audio standards. Programs playing on the JACK bus can route their audio I/O freely between one another, permitting complex scenarios, such as routing the output of a MIDI-controlled softsynth into a hard-disk recorder while applying modulated plugin effects, all in real time with low-latency. Though a relative newcomer to the Linux audio world, JACK already has caught the attention of many developers and users, and we are rapidly approaching the point where its deployment and use will be a matter of course for Linux audio programmers and normal users alike.

The Testing Environment

The testing hardware included a generic machine with an 800MHz AMD Duron CPU, 512MB RAM and a 15GB IDE hard disk. The audio hardware consisted of two sound cards, a SoundBlaster SBLive Value and a SoundBlaster PCI128; a Casio CZ101 synthesizer was used for external MIDI keyboard input. I used Steve Ratcliff's pmidi MIDI file player, and I also employed a second computer running Voyetra's Sequencer Plus Gold under MS-DOS. The video system included a generic 19" monitor and a Voodoo3 graphics card. Audio output from the sound cards ran to a Yamaha DMP7 mixer then out to a 100-watt QSC power amplifier and a pair of Yorkville Sound YSM-10 reference speakers.

The low- and middle-level software included Linux kernel 2.4.5 patched for low latency, the ALSA 0.9.0rc6 package (audio library, drivers and utilities), the latest JACK and the LADSPA plugin sets from Richard Furse and Steve Harris. Other support software included Maarten de Boer's alsamixergui and Bob Ham's ALSA MIDI patch bay, both of which provide GUIs (and more) for the ALSA alsamixer and aconnect utilities.

Running Softsynths as Root?

Many of the synths profiled here include the recommendation to be run with root permissions, either as root yourself or by making the binary suid root. Doing so usually ensures a higher priority for the running application but is also considered a serious security risk if the user is on a network.

Apart from the security issues, I should explain that when a real-time process runs away from the root user the outcome is not pretty, and your machine may lock up entirely. In one test while running as root user, simply specifying an unrecognized MIDI device froze my system. So be warned. Running as root can indeed enhance performance, but you also are running risks. Run normal applications as a normal user as much as possible.

The Linux Softsynth Roundup

The Software Synthesis section of the Linux Sound & Music Software site includes a subsection of Softsynths & Samplers. More than 30 URLs are currently active, taking you to a variety of software synthesizers. Table 1 has pruned some of that variety by focusing on synthesizers capable of polyphonic (plays many notes at once) real-time output, ignoring off-line synthesizers and environments such as Csound or RTCmix. Due to their real-time nature, I have included beatbox programs and the MAX-like environments of Pd and jMax.

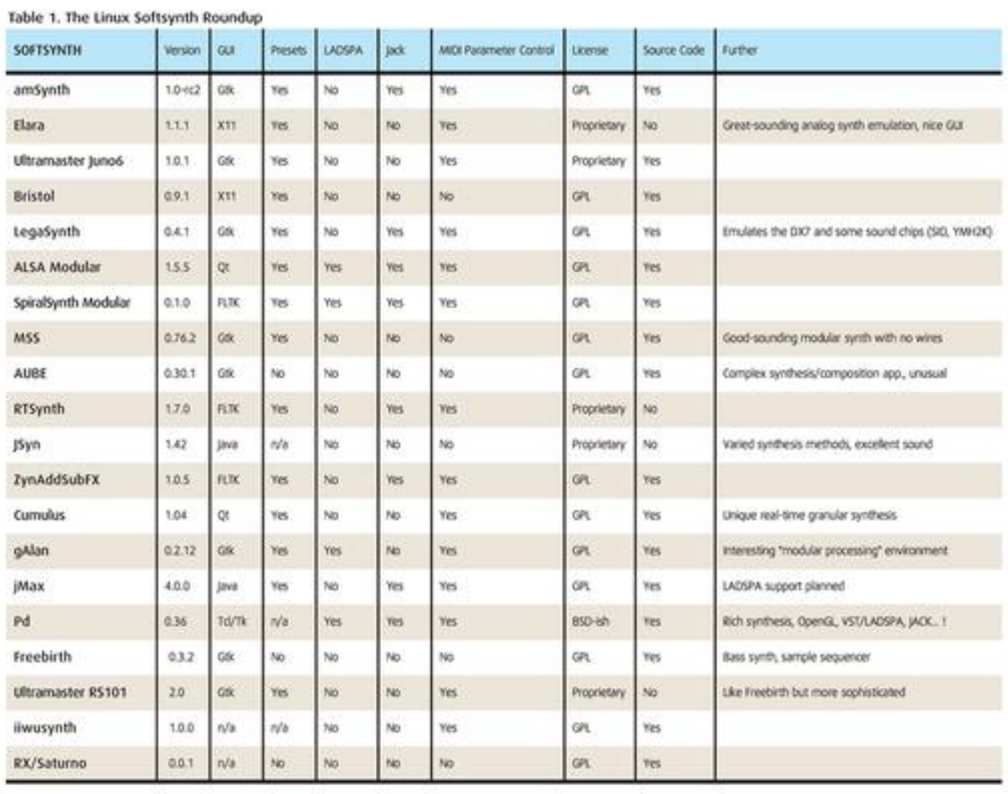


Table 1. The Linux Softsynth Roundup

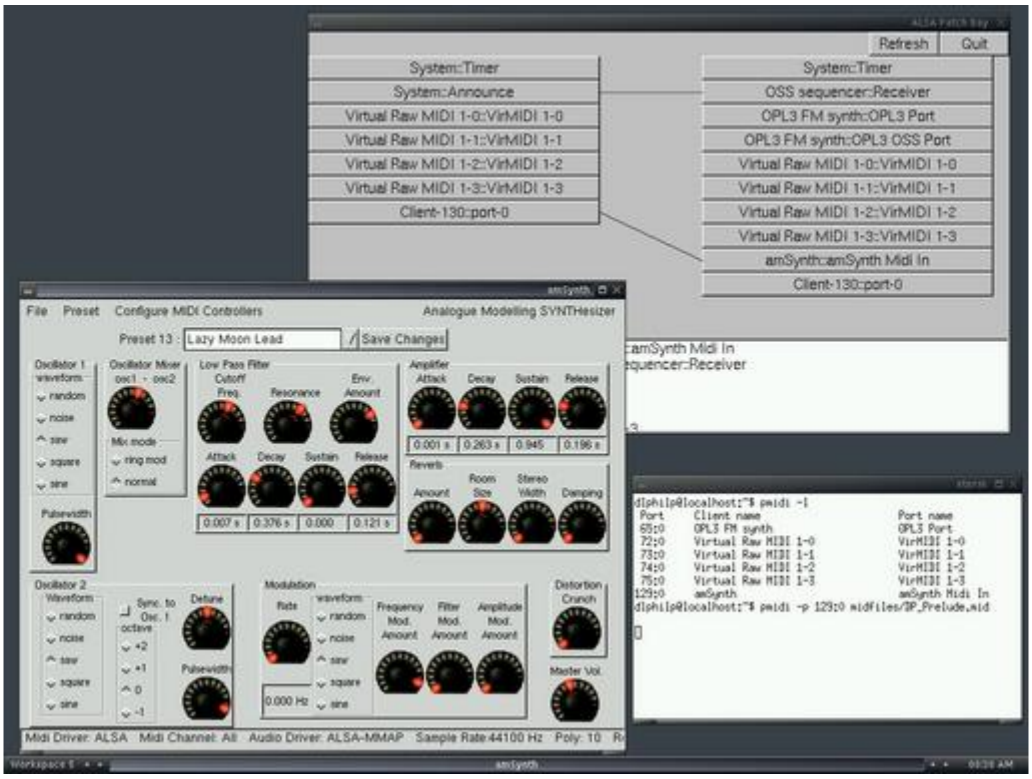
SOFTSYNTH	Version	GUI	Presets	LADSPA	Jack	MIDI Parameter Control	License	Source Code	Further
amSynth	1.0-rc2	Gtk	Yes	No	Yes	Yes	GPL	Yes	
Elara	1.1.1	X11	Yes	No	No	Yes	Proprietary	No	Great-sounding analog synth emulation, nice GUI
Ultramaster Juno6	1.0.1	Gtk	Yes	No	No	Yes	Proprietary	Yes	
Bristol	0.9.1	X11	Yes	No	No	No	GPL	Yes	
LegaSynth	0.4.1	Gtk	Yes	No	Yes	Yes	GPL	Yes	Emulates the DX7 and some sound chips (SID, YM2K)
ALSA Modular	1.5.5	Qt	Yes	Yes	Yes	Yes	GPL	Yes	
SpiralSynth Modular	0.1.0	FLTK	Yes	Yes	Yes	Yes	GPL	Yes	
MSS	0.76.2	Gtk	Yes	No	No	No	GPL	Yes	Good-sounding modular synth with no wires
AUBE	0.30.1	Gtk	No	No	No	No	GPL	Yes	Complex synthesis/composition app., unusual
RTSynth	1.7.0	FLTK	Yes	No	Yes	Yes	Proprietary	No	
JSyn	1.42	Java	n/a	No	No	No	Proprietary	No	Varied synthesis methods, excellent sound
ZynAddSubFX	1.0.5	FLTK	Yes	No	Yes	Yes	GPL	Yes	
Cumulus	1.04	Qt	Yes	No	No	Yes	GPL	Yes	Unique real-time granular synthesis
gAlan	0.2.12	Gtk	Yes	Yes	No	Yes	GPL	Yes	Interesting "modular processing" environment
jMax	4.0.0	Java	Yes	No	Yes	Yes	GPL	Yes	LADSPA support planned
Pd	0.36	Tcl/Tk	n/a	Yes	Yes	Yes	BSD-ish	Yes	Rich synthesis, OpenGL, VST/LADSPA, JACK... !
Freebirth	0.3.2	Gtk	No	No	No	No	GPL	Yes	Bass synth, sample sequencer
Ultramaster RS101	2.0	Gtk	Yes	No	No	Yes	Proprietary	No	Like Freebirth but more sophisticated
iiwusynth	1.0.0	n/a	n/a	No	No	Yes	GPL	Yes	
RX/Saturno	0.0.1	n/a	No	No	No	No	GPL	Yes	

Table 1. The Linux Softsynth Roundup

Because this article is a roundup and not a shoot-out, I'll profile only some selections from Table 1 and prune it a bit more. The profiles represent a cross section of the various synthesis methods, but I have purposely focused on the standalone softsynths.

amSynth

Nick Dowell's amSynth is an excellent representation of the dedicated subtractive synthesis architecture. The signal flow is fixed in a classic design. The output of the two oscillators is routed through a filter and an amplifier, then that signal can be given to the effects (amSynth provides reverb and distortion) and/or modulation stages for final processing before heading to your sound card digital-to-analog converter (DAC). In classic synthesis, the main parts of this design are referred to as the VCO (voltage-controlled oscillator), the VCF (voltage-controlled filter) and the VCA (voltage-controlled amplifier).


System::Timer
System::Announce
Virtual Raw MIDI 1-0::VirMIDI 1-0
Virtual Raw MIDI 1-1::VirMIDI 1-1
Virtual Raw MIDI 1-2::VirMIDI 1-2
Virtual Raw MIDI 1-3::VirMIDI 1-3
Client-130::port-0
OSS sequencer::Receiver
OPL3 FM synth::OPL3 Port
OPL3 FM synth::OPL3 OSS Port
amSynth::amSynth Midi In
Refresh
Quit
Analogue Modelling SYNTHesizer
Preset 13 : Lazy Moon Lead
Save Changes
Midi Driver: ALSA Midi Channel: All Audio Driver: ALSA-MMAP Sample Rate:44100 Hz Poly: 10

```
dlphilp@localhost:~$ pmidi -l
 Port     Client name                       Port name
 65:0     OPL3 FM synth                     OPL3 Port
 72:0     Virtual Raw MIDI 1-0              VirMIDI 1-0
 73:0     Virtual Raw MIDI 1-1              VirMIDI 1-1
 74:0     Virtual Raw MIDI 1-2              VirMIDI 1-2
 75:0     Virtual Raw MIDI 1-3              VirMIDI 1-3
129:0     amSynth                           amSynth Midi In
dlphilp@localhost:~$ pmidi -p 129:0 midfiles/JP_Prelude.mid
```

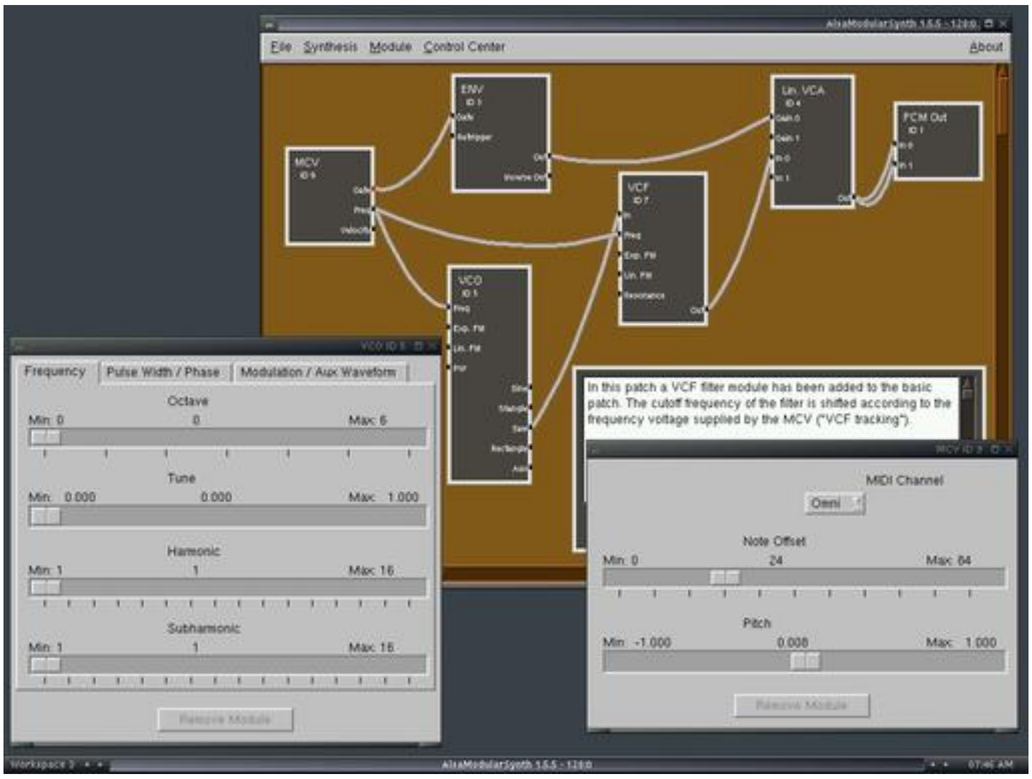
Figure 1. amSynth

amSynth includes some great-sounding presets. No General MIDI patch is set to support the General MIDI standardized map of instruments, but it does respond to incoming MIDI Program Change messages. Because amSynth is essentially a standalone single instrument that handles only one MIDI channel at a time, it is perhaps best used as a lead or pad instrument. It can be driven by a MIDI sequencer.

The full name for amSynth is Analogue Modelling Synthesizer. There are no real voltage-controlled components, so we might rightly wonder if Nick has been able to achieve his goal of modelling the sound of an analog subtractive synth. I'm happy to report that the sounds from amSynth are fat and lively, but you need not take my word for it. The excellent demos on the amSynth home page show off its sounds far better than I can describe here.

ALSA Modular Synth

Dr Matthias Nagorni has written a variety of useful applications and utilities for ALSA, JACK and LADSPA; however, his current crowning achievement must be his wonderful ALSA Modular Synthesizer (AMS). This software emulates the great modular synths of yesteryear, providing the user with a large selection of modules to choose from.



In this patch a VCF filter module has been added to the basic patch. The cutoff frequency of the filter is shifted according to the frequency voltage supplied by the MCV ("VCF tracking").

Octave Min: 0 0 Max: 6

Tune Min: 0.000 0.000 Max: 1.000

Harmonic Min: 1 1 Max: 16

Subharmonic Min: 1 1 Max: 16

MIDI Channel Omni

Note Offset Min: 0 24 Max: 84

Pitch Min: -1.000 0.000 Max: 1.000

Figure 2. ALSA Modular Synth (AMS)

Figure 2 illustrates AMS in its most basic form. In a subtractive synthesis patch, the routing essentially is identical to that used by amSynth, but the difference lies in the greater flexibility of AMS. Unlike the fixed design of amSynth, AMS is completely flexible with regard to the connections between its various modules.

Most modules cheerfully accept arbitrary input connections and have little or no concern for the destination of their own outputs. But beware; when modules are connected in atypical configurations, the output can be quite unusual or even overpowering, so take care with your system volume control when testing such patches. Each module has its own dialog (shown in Figure 2), which is opened by right-clicking over the module's name.

Dr Nagorni supplied the following informative notes:

> AMS implements special features to ensure that all three major synthesis methods [additive, subtractive, FM] can be easily used with it. The module Dynamic Waves implements additive synthesis of up to eight oscillators in one single module. Each harmonic can be shaped with an eight-point envelope, and the envelopes are graphically visualized. To enable easy access to integer harmonic tuning, useful for FM, VCOs have an additional harmonic and subharmonic slider. There is also the required linear FM input port. For subtractive synthesis to work properly, it is crucial that control voltages obey the classical logarithmic convention of 1V/Octave. This way, you can move the

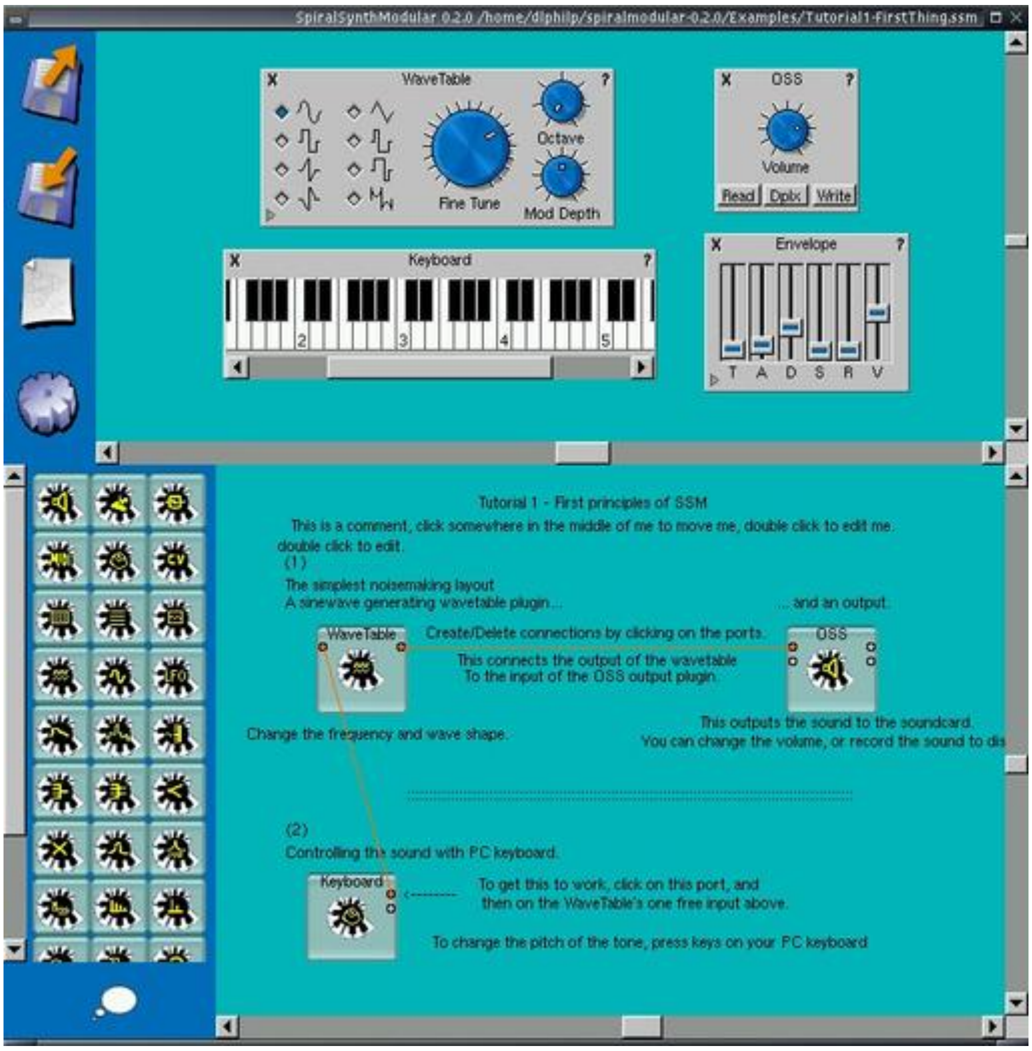
> filter cutoff wherever you like, and you can still have perfect VCF tracking. Log Frequency is also useful at other places, including vibrato with an LFO.

AMS has been designed for real-time work. It is especially well-suited for MIDI control, and most parameters can be linked to a MIDI controller for real-time changes. AMS can be used equally well as a monophonic or polyphonic synth, and multiple instances of AMS may communicate over JACK to create a multi-timbral setup. Its support for the LADSPA plugins extends its already rich feature set, making AMS an ideal solution for those of us without access to a hardware synthesizer. A complete MIDI composition environment can be built from nothing more than a reasonably fast machine, one of the fine Linux MIDI sequencers, such as MusE or Rosegarden, and AMS.

Some setups will work better than others, so the good doctor has prepared a healthy supply of sample patches for your study and experimentation. You can hear some of them in the demo files available from the AMS home page, but as with all the synths profiled here, I suggest you download and install it yourself to see and hear what really it can do.

SpiralSynth Modular

First there was SpiralSynth, then there was the SpiralLoops program, a cool looping sequencer, and then there was the SpiralSynthBaby, designed to be a plugin for SpiralLoops. Finally, developer Dave Griffiths decided to roll all of them into one open-ended modular synthesizer construction kit called SpiralSynth Modular (SSM). Like AMS, SpiralSynth Modular provides the user with a canvas and a palette of modules for placement and connection on the canvas, but SSM has its own unique design and sound-producing capabilities.



Tutorial 1 - First principles of SSM

This is a comment, click somewhere in the middle of me to move me, double click to edit me.
double click to edit.

(1)
The simplest noisemaking layout
A sinewave generating wavetable plugin... ...and an output.

Create/Delete connections by clicking on the ports.

This connects the output of the wavetable
To the input of the OSS output plugin.

Change the frequency and wave shape.

This outputs the sound to the soundcard.
You can change the volume, or record the sound to dis

(2)
Controlling the sound with PC keyboard.

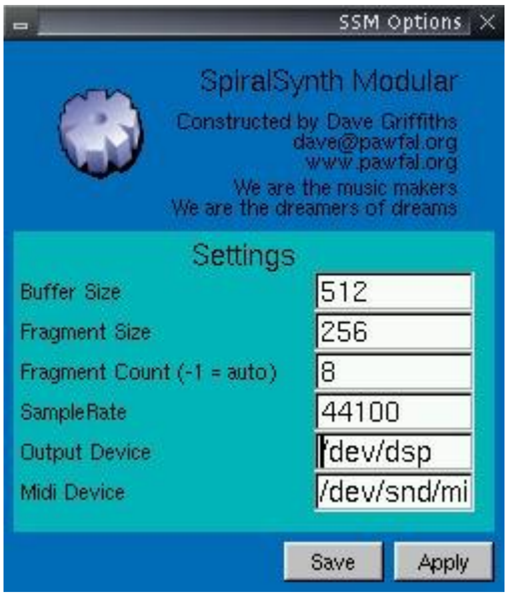
<---------- To get this to work, click on this port, and
then on the WaveTable's one free input above.

To change the pitch of the tone, press keys on your PC keyboard

Figure 3. SpiralSynth Modular

Figure 3 shows off SSM running its first tutorial patch. This example shows a simple type of synthesis called wavetable synthesis. The wavetable is a predefined stored waveform (sine, square, triangle, pulse, etc.) that is triggered by the virtual keyboard and modified by the envelope generator before heading out to the sound card DAC through the OSS output module. In the example, we can see that the synthesizer is played through the computer keyboard, but SSM also provides a MIDI module for receiving and routing MIDI messages. The keyboard module is a nice touch, and I had great fun with it playing SSM from my laptop's QWERTY keys.

SSM does not function as a native ALSA sequencer client, so it cannot be wired directly to an ALSA port like amSynth or AMS. However, it can be hooked to the standard OSS/Free MIDI device (/dev/midi) for input from any hardware or software connected to that device. If your machine has no MIDI hardware, you can employ ALSA's virmidi virtual MIDI ports by setting the MIDI channel to the appropriate port in the SSM Options (/dev/snd/midiC1D0 for my laptop; see Figure 4). This enables connection to other ALSA-aware processes through aconnect or the ALSA patch bay.


SSM Options

SpiralSynth Modular

Constructed by Dave Griffiths
dave@pawfal.org
www.pawfal.org

We are the music makers
We are the dreamers of dreams

Settings

Buffer Size	512
Fragment Size	256
Fragment Count (-1 = auto)	8
SampleRate	44100
Output Device	/dev/dsp
Midi Device	/dev/snd/mi

Figure 4. SSM Options

Dave Griffiths has thoughtfully provided some excellent demos of the synth on the SSM home site. Its FLTK interface is pleasant and easy to use, the program includes a generous share of interesting and useful modules, including LADSPA support, and the latest version can be built with support for JACK. Dave plans to include a much-improved plugin version of SpiralLoops for a soon-to-arrive version of SSM, and we can expect more direct support for ALSA as well.

RTSynth

Stefan Nitschke's RTSynth is one of my favorite softsynths. It is another excellent example of the patcher synth. A main canvas is presented, iconic modules are deposited and connected together on the canvas, and right-clicking on a module opens a dialog for editing its parameters. RTSynth is the only softsynth represented here that creates its sounds via physical modelling.

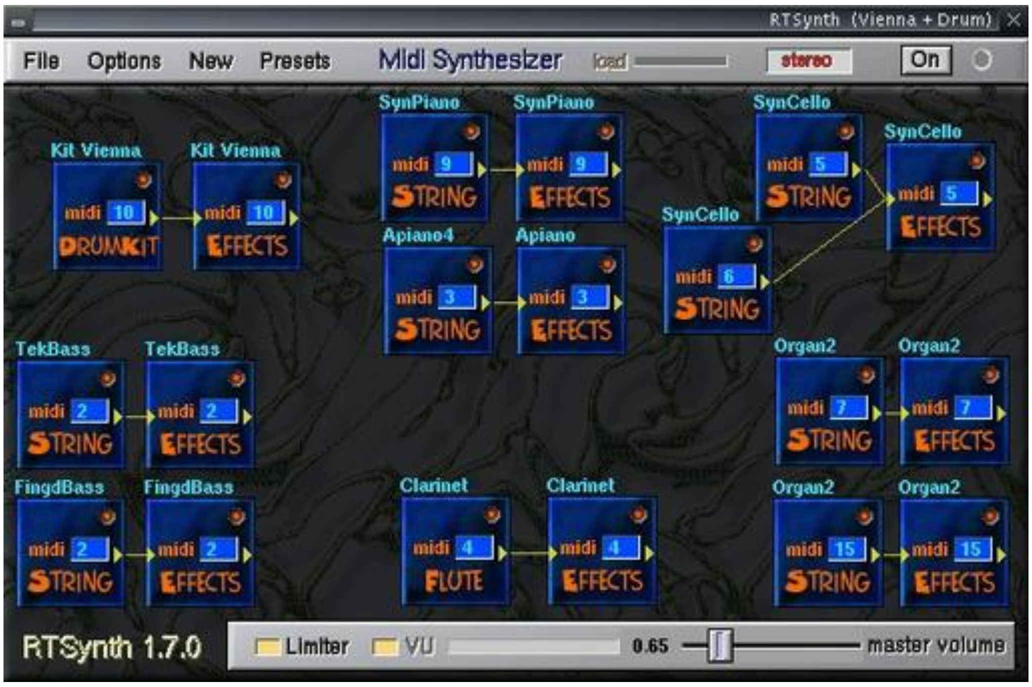


Figure 5. RTSynth

Physical modelling synthesis is capable of extremely realistic sounds. Some of RTSynth's patches are quite convincing. The examples on the RTSynth home page show off some amazing acoustic and electric guitar sounds in full arrangements with bass, drums and keyboards. RTSynth is a multi-timbral softsynth, complete with drumkits and effects processing, and the demos really showcase its capabilities as a single-solution softsynth.

RTSynth is ALSA- and JACK-aware. It is fully MIDI-capable under ALSA and the older OSS/Free kernel sound modules. On systems lacking the ALSA drivers, it is still possible to connect RTSynth to external processes, such as a concurrently running MIDI sequencer via the UNIX mechanism known as a named pipe. A named pipe provides an easy method of interprocess communication for programs that may have no other way to share data. Using RTSynth as an example, here's how you set up a named pipe.

First, create the pipe with the mkfifo utility:

```
mkfifo $HOME/tmp/midififo
```

Next, prepare RTSynth for receiving data from the pipe:

```
RTSynth < $HOME/tmp/midififo
```

Finally, you must indicate the named pipe as the preferred output device for the driving application. In the following example, I've used Simon Kagedal's clavier virtual keyboard:

```
clavier -o $HOME/tmp/midififo
```

Now you can play RTSynth directly from the virtual keyboard. You also can use a normal, unnamed pipe to route the output from a process into RTSynth using this type of command:

```
cat foo | RTSynth
```

These connectivity strategies are particularly effective in the absence of MIDI hardware and/or the ALSA virmidi driver.

Bristol

Nick Copeland is perhaps best known for his SLab hard-disk recording system, but he also has given us the Bristol Synthesizer Emulator. This softsynth provides GUIs and synthesis engines for emulations of the Mini Moog, Moog Voyager, Sequential Circuits Prophet-5, Roland Juno-6 and Yamaha DX7 synthesizers. It also provides graphic interfaces and engines for the Hammond B3 and Vox Continental organs and the Fender Rhodes electric piano. Bristol even emulates a generic mixing board and the Yamaha Pro10 digital mixer, but they were not tested for this review.

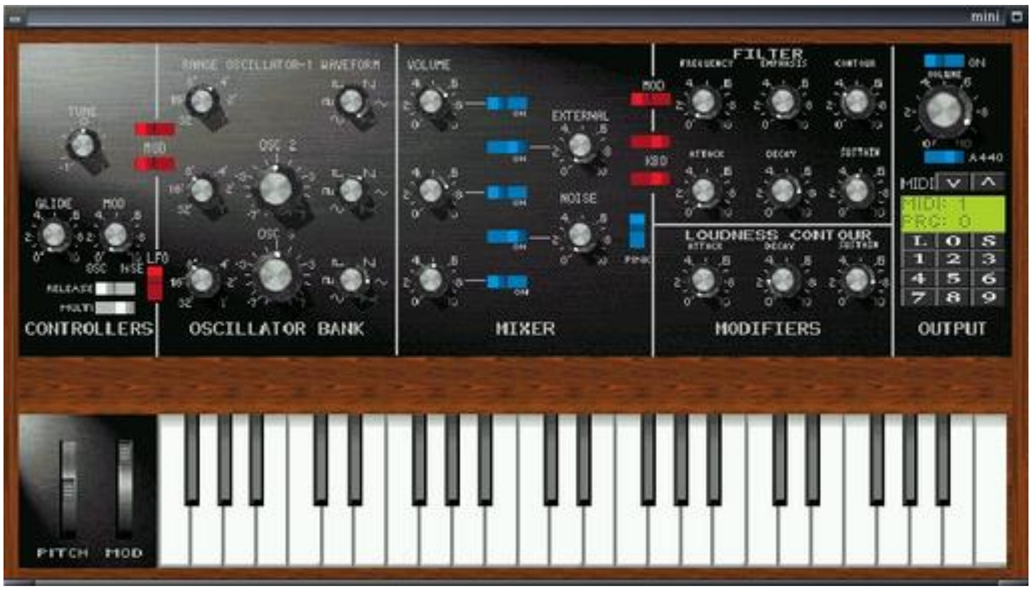


Figure 6. Bristol

As shown in Figure 6, the GUIs are nicely drawn, but they are more than mere eye candy. Nick has emulated the controls and functions found on the original synths as much as possible; however, not all of a particular synth's features may be implemented yet, and Nick notes that some emulations (notably the DX7) still need some work. Meanwhile, all those switches and knobs and wheels can be flipped, twirled and rotated in real time with smooth response and fast parameter updates. Bristol accomplishes a rather daunting task by providing not only the look-alike graphics for its variety of synthesizers and keyboards but the sound-alike synthesis engines as well.

Running Bristol with **./startBristol -v -h** lists the runtime options to give the synth a wide degree of performance customization. For example, I started Bristol with **./startBristol -alsa -seq -bufsize 2048 -voices 6**, which launches Bristol in its default Mini Moog mode, declares ALSA as the driver source, registers Bristol with the ALSA sequencer, sets the sound-card buffer size (the default value is 1,024, but Nick recommends 2,048 for my SBLive) and restricts the polyphony to six voices (Bristol's default polyphony is 16 voices). Incidentally, Bristol can be run in multiple instances with simultaneous control, effectively letting you layer synths exactly like we did in the old days.

I would need much more space to describe each of Bristol's interfaces adequately. The example I've placed at www.linux-sound.org/sounds demonstrates only the Mini Moog emulation, but it should give you an idea of what you can expect from this synth engine—some old-school synthesizer fun.

Ultramaster Juno6

This synth is an excellent example of an emulated hardware synthesizer. The Juno6 keyboard and panel controls are faithfully rendered, and like the Bristol synth, all controls are active and available for manipulation at any time. I've owned a Juno6, and Ultramaster's audio emulation is quite faithful to the original, but with the stability of intonation of a digital synth. Best of all, the arpeggiator works. Those of us who remember such amenities probably will have great fun with this feature; alas, arpeggiators are not so common anymore, so newbies can expect to while away many an hour finding interesting and fun uses for this function.

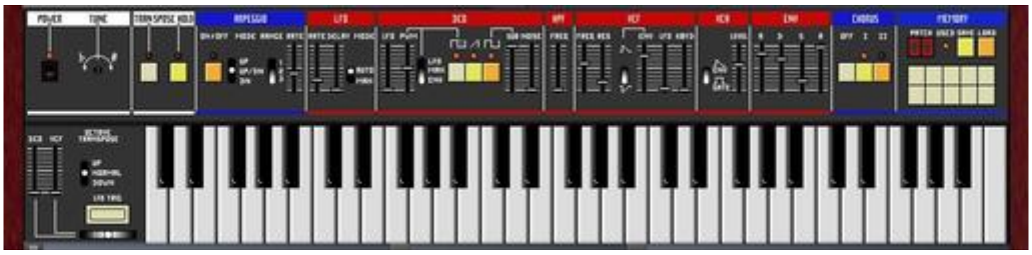
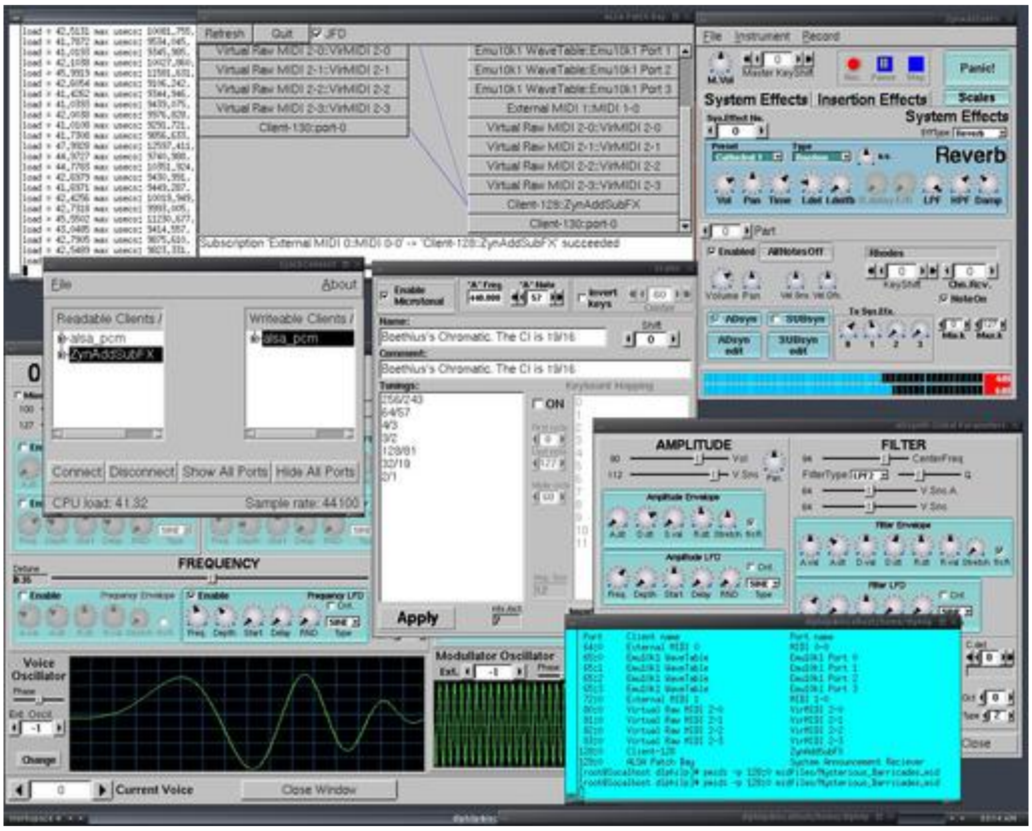


Figure 7. Ultramaster Juno6

The Juno6 is a straightforward implementation of subtractive synthesis, lending itself to sounds with dramatic filter sweeps. A short example WAV file can be found on the Ultramaster home page, but you'll learn more about the synth's sound and capabilities simply by playing around with it.

ZynAddSubFX

Paul Nasca's ZynAddSubFX is an interesting hybrid of additive and subtractive synthesis, with an added effects section for further processing. If that's all ZynAddSubFX offered, it still would grab your attention. An excellent FLTK interface invites experimentation with the various parameters of the synthesis strategies, and as an ALSA-aware client, you can drive the synth from your favorite MIDI sequencer. Figure 8 shows ZynAddSubFX working with the pmidi MIDI file player. It also shows ZynAddSubFX's Scales dialog opened to a collection of tunings from the Scala program. Selecting a new scale automatically updates the current patch's tuning, which invites exploration of unusual intonations and induces some interesting changes upon familiar material.



Virtual Raw MIDI 2-0:VirMIDI 2-0
Virtual Raw MIDI 2-1:VirMIDI 2-1
Virtual Raw MIDI 2-2:VirMIDI 2-2
Virtual Raw MIDI 2-3:VirMIDI 2-3
Client-130:port-0

Emu10k1 WaveTable:Emu10k1 Port 1
Emu10k1 WaveTable:Emu10k1 Port 2
Emu10k1 WaveTable:Emu10k1 Port 3
External MIDI 1:MIDI 1-0
Virtual Raw MIDI 2-0:VirMIDI 2-0
Virtual Raw MIDI 2-1:VirMIDI 2-1
Virtual Raw MIDI 2-2:VirMIDI 2-2
Virtual Raw MIDI 2-3:VirMIDI 2-3
Client-128:ZynAddSubFX
Client-130:port-0

Subscription 'External MIDI 0:MIDI 0-0' -> 'Client-128:ZynAddSubFX' succeeded

Name: Boethius's Chromatic. The CI is 19/16

Comment: Boethius's Chromatic. The CI is 19/16

Tunings:
256/243
64/57
4/3
3/2
128/81
32/19
2/1

```
Port     Client name                  Port name
64:0     External MIDI 0              MIDI 0-0
65:0     Emu10k1 WaveTable            Emu10k1 Port 0
65:1     Emu10k1 WaveTable            Emu10k1 Port 1
65:2     Emu10k1 WaveTable            Emu10k1 Port 2
65:3     Emu10k1 WaveTable            Emu10k1 Port 3
72:0     External MIDI 1              MIDI 1-0
80:0     Virtual Raw MIDI 2-0         VirMIDI 2-0
81:0     Virtual Raw MIDI 2-1         VirMIDI 2-1
82:0     Virtual Raw MIDI 2-2         VirMIDI 2-2
83:0     Virtual Raw MIDI 2-3         VirMIDI 2-3
128:0    Client-128                   ZynAddSubFX
129:0    ALSA Patch Bay               System Announcement Reciever
[root@localhost dlphilp]# pmidi -p 128:0 midfiles/Mysterious_Barricades.mid
[root@localhost dlphilp]# pmidi -p 128:0 midfiles/Mysterious_Barricades.mid
```

Figure 8. ZynAddSubFX

ZynAddSubFX is multi-timbral, with a different instrument per MIDI channel, making it another good choice for an all-purpose softsynth (minus drums, alas). Its sounds are created by straightforward synthesis methods, but the deployment of those methods and the program's excellent interface combine to help make some fine sounds. Performances can be recorded directly within ZynAddSubFX, and the developer has placed several demos on-line that depict the synth's power as a standalone multi-timbral softsynth. ZynAddSubFX is the newest softsynth profiled here, but its development is steady. As this article was being written, I learned that ZynAddSubFX is now JACK-aware (Figure 8), so with support for scales and tunings from Scala, the ALSA sequencer client configuration and JACK connectivity, this synth is a fine representative example of modern Linux audio software.

jMax and Csound

I know I promised to steer clear of profiling the more language-based synthesis environment, but I also mentioned the blurring tendency occurring with developments in those environments. jMax rapidly is evolving into a rich multimedia composition/processing suite, but it also can be utilized as a straightforward SWSS toolkit. Figure 9 illustrates a simple jMax synthesis patch, complete with self-documentation. Although this example is itself trivial, jMax is capable of far more complex synthesis patches.

Figure 10 demonstrates Istvan Varga's csoundfltk (a Csound package optimized for Linux) running ImproSculpt, a real-time sampler with a rather complex FLTK graphic interface. This example is not really a synthesis patch, but it shows off the power of the Csound FLTK widget set that lets users design GUI panels and control systems for their Csound synthesis and processing designs. Other examples demonstrate Csound as a straightforward synthesizer, and interested readers should check out the material available at www.csounds.com for more examples of the FLTK/Csound powerhouse.

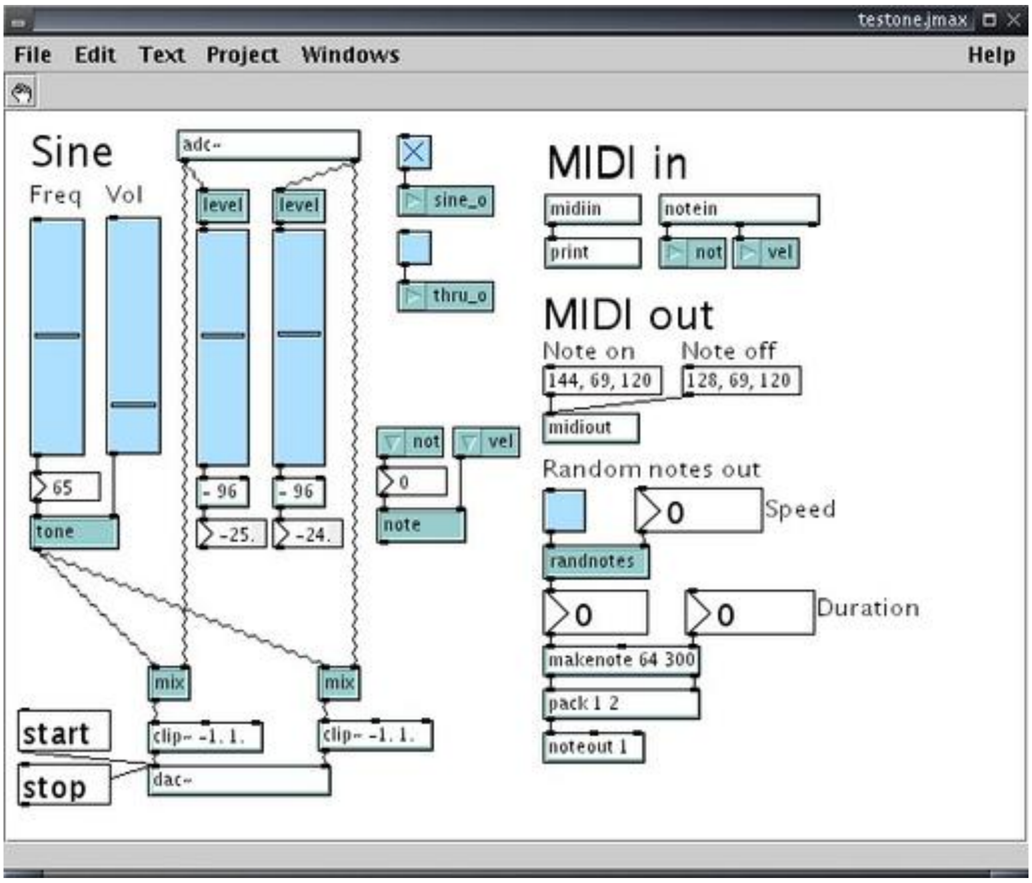


Figure 9. jMax

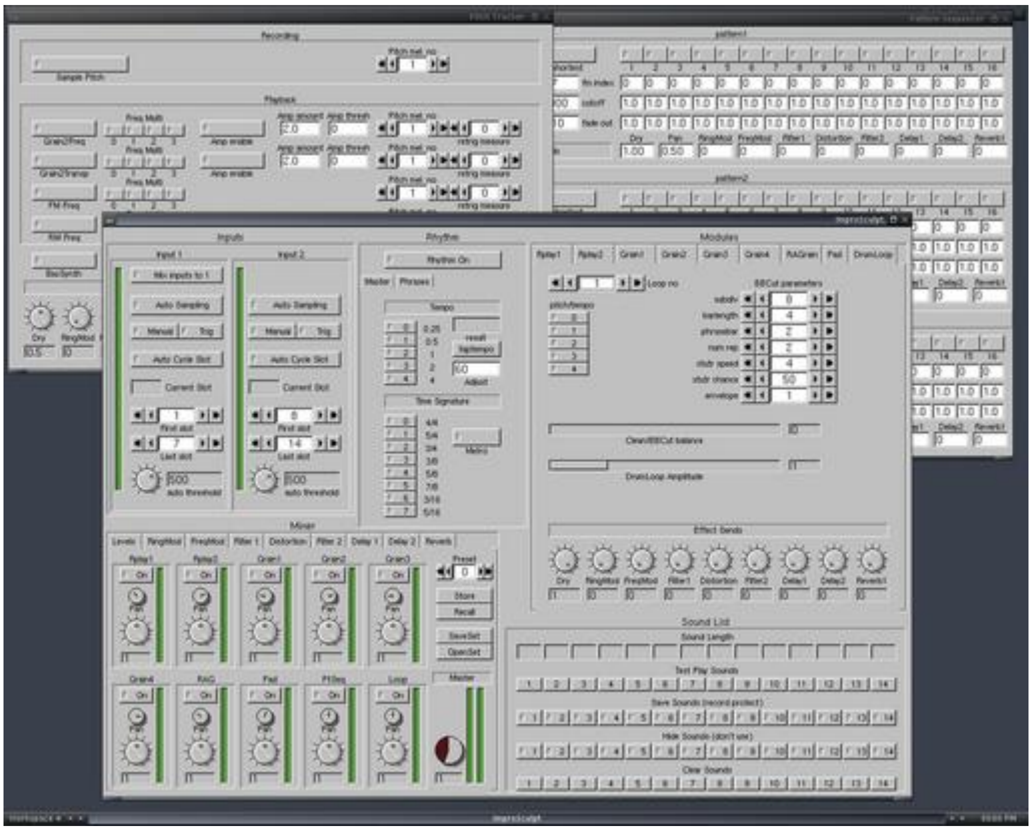


Figure 10. Istvan Varga's csoundfltk running Oeyvind Brandtsegg's ImproSculpt

The Wrap

The best closing advice I can give is to suggest that you check out some of the profiled softsynths yourself. I can wax enthusiastic here in print, but the real proof is in the hearing. So go forth, download and install some of this software, and make some joyful noises. I'll be listening.

Resources

email: dlphilp@bright.net

Dave Phillips is a musician, teacher and writer living in Findlay, Ohio. He has been an active member of the Linux audio community since his first contact with Linux in 1995. He is the author of The Book of Linux Music & Sound, as well as numerous articles in *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Regular Expressions

Giovanni Organtini

Issue #109, May 2003

For precision of text manipulation and description, it's hard to beat the power of regexps.

Imagine you are looking for a name in a telephone directory, but you can't remember its exact spelling. You can spend ages searching through all the possible combinations, unless you have a tool that extracts the relatively small number of options that matches your search, however incomplete it may be. Regular expressions are such a tool.

Roughly speaking, a regular expression (or regexp) is a string that describes another string or a group of strings. Several applications can profit from this ability: Perl, sed, awk, egrep and even Emacs (try Ctrl-Alt-% after reading this article) to name a few.

In fact, many of you already have used some sort of regular expression. In the shell command:

```
ls *.pl
```

the characters *.pl act as a regular expression. That is, it is a string that describes all the strings composed by any number of characters of any kind (*), followed by a period (.), followed by two given characters (pl).

The standard set of rules used for composing regular expressions is able to describe all strings, no matter how complicated they are. Unfortunately, life is always more complicated. It turns out that at least two different versions of regular expressions exist: extended and basic. Moreover, not all applications support all the possible rules.

Basics of Regular Expressions

A regular expression is said to *match* a given string if it correctly describes it. A given regular expression can match with zero to many strings. By convention, regular expressions are written between slashes (/.../). In what follows, I use extended regular expressions.

The simplest regular expression is a plain alphanumeric string. Such a regexp matches with all strings containing its content as a substring. As an example, consider the following verse from *Cenerentola*, my favorite opera by G. Rossini: “Zitto, zitto, piano, piano, senza strepito e rumore.” The regexp /piano/ is said to match with the verse, because the latter contains the same characters, with the same sequence, of the regexp.

In order to better understand the examples I discuss, you can play with the following Perl script, trying variations of the regexp it contains:

```
#!/usr/bin/perl
$verse = "Zitto, zitto, piano, piano, senza " .
         "strepito e rumore";
if ($verse =~ /piano/) {
   print "Match!\n";
} else {
   print "Do not match!\n";
}
```

In Perl, the operator =~ compares two regular expressions and returns “true” if they match.

A few characters (called metacharacters) are not recognized as ordinary characters and are used for special purposes. The *, for example, is used to match zero or more times a group of characters that, in turn, is identified by a couple of parentheses defining an atom, or a group of characters that must be considered as a single entity. The regexp **/(piano,)*/** matches with the sample verse because the characters “ **piano,**”, forming an atom, are repeated twice. If the atom is composed of a single character, parentheses may be omitted.

The meaning of the * within a regular expression is different from the one it has in the shell. In regular expressions, the * is a modifier; it describes the multiplicity of the atom on its left. So, the string “piano” is matched by **p*** in a shell, but not within a regular expression: **/p*/** matches with p, pp, ppp and so on, and even with a null string.

To specify that an atom's multiplicity ranges between N and M, the symbol {N,M} is used. {N} matches strings with exactly N repetitions of the preceding atom; {N,} will match at least N of them. So, the following regular expressions will match:

```
/( piano,){0,10}/
/( piano,){1,2}/
/( piano,){2}/
```

Of course, the first regexp will match with “ piano, piano, piano” too.

The metacharacters + and ? are shorthands, respectively, for {1,} and {0,1}.

Matched parenthesized atoms are automatically stored into special variables (called back references) identified by the symbol \ followed by a number. The first parenthesized atom occurrence in a regular expression will be stored in \1, the second in \2 and so on. For example:

```
/Z(itto), z\1, ( piano,)\2/
```

will match the above-mentioned verse (imagine that \1 = “itto” and \2 = “piano,”).

The . metacharacter can describe any character, so the regular expression **/.(itto), .\1/** matches both “Zitto, zitto” and “zitto, zitto”. However, it even matches with “Ritto, ritto”, which does not have the same meaning. To avoid being so generic, you can specify a set of possible alternatives, listing the possible characters in brackets:

```
/[Zz](itto), [Zz]\1/
```

A dash in brackets is used to specify a range of characters. For example, **/[a-z]/** matches all lowercase characters, and **/[A-Z]/** matches all uppercase characters. **/[a-zA-Z0-9_]/** matches any alphanumeric character or an undesrcore.

The metacharacter | can be used to express different alternatives. It works like a logical OR statement. Therefore:

```
/Zitto|zitto/
```

will match with both “Zitto” and “zitto”.

The metacharacters ^ and $ match, respectively, the beginning and the end of a string. If used inside brackets, the caret is interpreted as the negation operator. So:

```
/[^a-z]itto/
```

will match Zitto, but not zitto (**[^a-z]** can be read as “any letter that is not a lowercase letter”).

To match a metacharacter it's enough to put a backslash (\) in front of it to tell the regexp to interpret it as an ordinary character. The \ character is often called an escape character.

Using Regular Expressions

To appreciate the power of regular expressions, let's look at a simple Perl script that helps system administrators look for authentication failures. For the following examples I used rather expressive regular expressions to show different features. You may write simpler ones to describe the same strings.

Each time someone fails to log in, syslogd writes messages to /var/log/messages that read like this:

```
Jul 26 16:35:25 myhost su(pam_unix)[2549]:
authentication failure; logname=verdi uid=500
euid=0
tty= ruser=organtin rhost=  user=root
Jul 27 14:54:36 myhost login(pam_unix)[688]:
authentication failure; logname=LOGIN uid=0
euid=0 tty=tty1 ruser= rhost=  user=mozart
```

These lines list the time at which the login attempt was made, the user who tried to log in as another user, if available, and the target user. For example, the user verdi tried to log in as root two times, while someone failed to log in as mozart from the console.

Consider the Perl script shown in Listing 1. It reads the /var/log/messages file, then identifies the lines that look interesting and extracts only the relevant information.

<u>Listing 1. Sample Perl Script for Finding Authentication Errors</u>

First of all, we select only relevant lines and match them with the regular expression /authentication failure/ shown on line 7. Everything else is discarded. Then each line is matched with a regular expression (line 8) that should be read as follows: take all the strings starting (^) with exactly three ({3}) alphabetic ([a-zA-Z]) characters, followed by a space, followed by at most two (+) characters that could be either numeric (0-9, equivalent in Perl to the metacharacter \d) or a space. After a space, an arbitrary number (*) of digits or semicolons must follow. The portion of the string described so far is enclosed in parentheses, so it is stored in a back reference called \1 (it is the first one). After that, any number of characters (.*) can be found before the string “logname=”. That string must be followed by any number of alphanumeric characters. Again, because there are a couple of parentheses, we will store them in \2. Any number of characters, finally, can be present before the string “user=”, followed by any number of alphanumeric characters. This all gets stored into \3.

From this example, you can see how it is possible to extract substrings from strings. You do not need to know their relative positions, as long as you can describe their appearance.

Perl provides a helpful feature for working with regexps. The automagic definition of Perl variables named after the back references as $1, $2 and so on, can be used after a regular expression has been matched. Perl also lets users define useful symbols, such as \d or \w (equivalent to [A-Za-z0-9_]), as well as POSIX-compliant symbols representing the same things (see **man perlre** for more information).

Basic Regular Expressions

Basic regular expressions are used by several other programs, like sed or egrep.

In basic regular expressions, the metacharacters |, + and ? do not exist, and parentheses and braces need to be escaped to be interpreted as metacharacters. The ^, $ and * metacharacters follow more complicated rules (see **man 7 regex** for more details). In most cases, however, they behave like their extended counterparts. It is often convenient to express the regular expression in the extended format, then add the escape characters when needed.

As an example, the script shown in Listing 2 generates an HTML-formatted page to read the content of system log files using an internet browser. Besides echoing HTML tags for the headers of the page and for a table, it simply lists files in a given directory and pipes the result to sed, which transforms it using a regexp. The syntax used by sed for text substitution is rather common and is something like:

```
s/regexp/replacement/
```

where regexp is a regular expression that must be replaced.

Listing 2. Example Script for Generating and HTML-Formatted Page for Reading Log Files

Essentially, the syntax represents a string composed of nine elements properly described by the appropriate regular expressions. For example **[rwxds-]** asks for the possible characters that can be found within the first element.

The latter part of the string consists of alphanumeric characters, with slashes interspersed. You may notice that the regular expression used in this case is (.*\/)(.*). The first group matches all characters preceding a (escaped) slash, i.e., the path name. The second group lists all the following characters (the filename). The number of slashes in the path doesn't matter. Regular expressions (both basic and extended), in fact, are said to be greedy—they try to match as many characters as possible.

The result of the script is written to standard output and can be redirected to a given file (by cron at fixed intervals, for example) to be shown on the Web.

Conclusion

Regular expressions are by far the most powerful tool for text manipulation and description, and they are well supported under Linux on many applications. Unfortunately, they are not supported at all (to my knowledge) by the most popular search engines because of their complexities. But, can you imagine how precise your search would be if you had the ability to describe the page you are looking for with a regular expression?

email: Giovanni.Organtini@roma1.infn.it

Giovanni Organtini (g.organtini@roma1.infn.it) is a professor of Introduction to Computing and Programming for Physicists at the University of Rome. He has used Linux for years, both for fun and at work, where it is used for the simulation of the CMS experiment (cmsdoc.cern.ch) on large farms and as part of a complex data-acquisition system and machine control. Before the birth of his son, Lorenzo, he used to travel, seeking good restaurants and attending concerts and operas.

Archive Index Issue Table of Contents

Advanced search

Advanced Memory Allocation

Gianluca Insolvibile

Issue #109, May 2003

Call some useful fuctions of the GNU C library to save precious memory and to find nasty bugs.

Dealing with dynamic memory traditionally has been one of the most awkward issues of C and C++ programming. It is not surprising that some supposedly easier languages, such as Java, have introduced garbage collection mechanisms that relieve programmers of this burden. But for hard-core C programmers, the GNU C library contains some tools that allow them to tune, check and track the usage of memory.

Memory Management Basics

A process' memory usually is classified as either static, the size is predetermined at compile time, or dynamic, space is allocated as needed at runtime. The latter, in turn, is divided into heap space, where malloc()'d memory comes from, and stack, where functions' temporary work space is placed. As Figure 1 shows, heap space grows upward, whereas stack space grows downward.

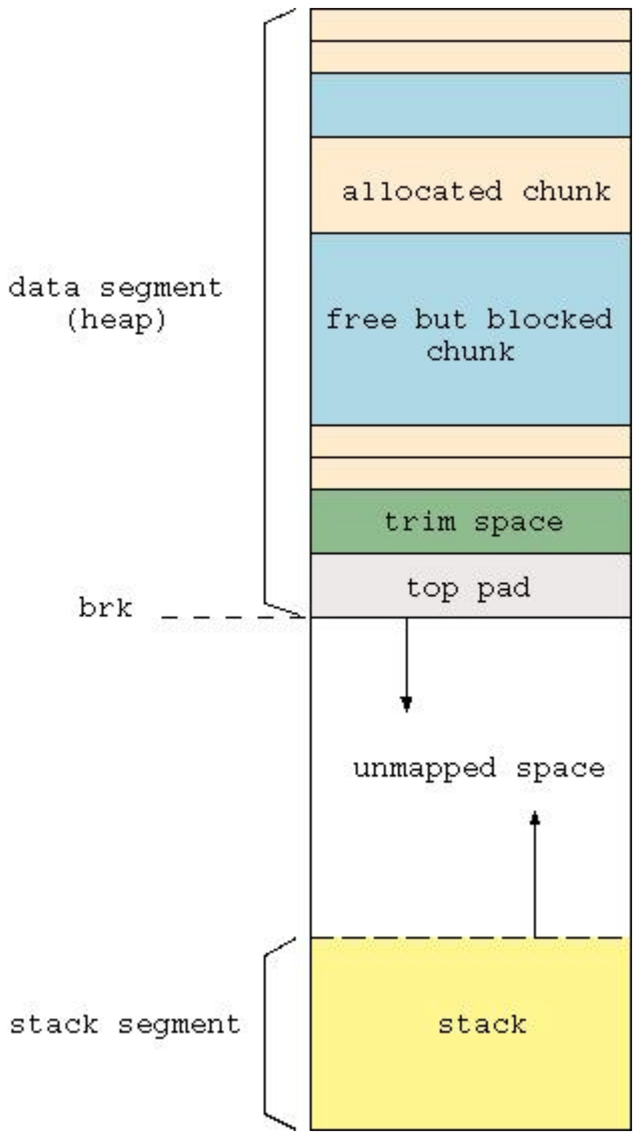


Figure 1. The heap and stack grow toward each other.

When a process needs memory, some room is created by moving the upper bound of the heap forward, using the brk() or sbrk() system calls. Because a system call is expensive in terms of CPU usage, a better strategy is to call brk() to grab a large chunk of memory and then split it as needed to get smaller chunks. This is exactly what malloc() does. It aggregates a lot of smaller malloc() requests into fewer large brk() calls. Doing so yields a significant performance improvement. The malloc() call itself is much less expensive than brk(), because it is a library call, not a system call. Symmetric behavior is adopted when memory is freed by the process. Memory blocks are not immediately returned to the system, which would require a new brk() call with a negative argument. Instead, the C library aggregates them until a sufficiently large, contiguous chunk can be freed at once.

For very large requests, malloc() uses the mmap() system call to find addressable memory space. This process helps reduce the negative effects of memory fragmentation when large blocks of memory are freed but locked by smaller, more recently allocated blocks lying between them and the end of the allocated space. In this case, in fact, had the block been allocated with brk(), it would have remained unusable by the system even if the process freed it.

Library functions that deal with dynamic memory are not limited to malloc() and free(), although these are by far the most-used calls. Other available functions include realloc(), to resize an already allocated block; calloc(), to allocate a cleared block; and memalign(), posix_memalign() and valloc(), to allocate an aligned block.

Dealing with Memory Status

The strategy adopted by the C library memory management code is optimized for generic memory usage profiles. Although this strategy produces good performance in most cases, some programs might benefit from slightly different parameter tuning. First, check your memory usage statistics by using either the malloc_stats() or the mallinfo() library calls. The former prints as a standard error a brief summary of memory usage in the program. This summary includes how many bytes have been allocated from the system, gathered with brk(); how many are actually in use, found with malloc(); and how much memory has been claimed, using mmap(). Here is a sample output:

```
Arena 0:
system bytes     =     205892
in use bytes     =     101188
Total (incl. mmap):
system bytes     =     205892
in use bytes     =     101188
max mmap regions =          0
max mmap bytes   =          0
```

If you need to have more precise information and want to make more than a printout, mallinfo() is helpful. This function returns a struct mallinfo containing various memory-related status indicators; the most interesting are summarized in the Sidebar “Useful Parameters Provided by mallinfo”. For a complete description of the structure, take a look at /usr/include/malloc.h.

Useful Parameters Provided by mallinfo()

Another useful function provided by libc is malloc_usable_size(), which returns the number of bytes you actually can use in a previously allocated memory block. This value may be more than the amount you originally requested, due to alignment and minimum size constraints. For example, if you allocate 30 bytes, the usable size is actually 36. This means you could write up to 36 bytes to that memory block without overwriting other blocks. This is an extremely awful and version-dependent programming practice, however, so please don't do it. The most useful application of malloc_usable_size() probably is as a debug tool. For example, it can check the size of a memory block passed from outside before writing to it.

Controlling the Allocation Strategy

You can alter the behavior of the memory management functions by adjusting some of the parameters exposed by the mallopt() function (Listings 1 and 2).

Listing 1. Setting the Trim Threshold with mallopt()

Listing 2. A smaller trim threshold might save space.

The prototype of this function and a basic set of four parameters are part of the SVID/XPG/ANSI standard. The current GNU C library implementation (version 2.3.1 as of this writing) honors only one of them (M_MXFAST), leaving three out. On the other hand, the library provides four additional parameters not specified by the standard. Tunable parameters accepted by mallopt() are described in the Sidebar “Tunable Parameters for mallopt()”.

Tunable Paramenter for mallopt()

Allocation tuning is possible even without introducing mallopt() calls inside your program and recompiling it. This may be useful if you want to test values quickly or if you don't have the sources. All you have to do is set the appropriate environment variable before running the application. Table 1 shows the mapping between mallopt() parameters and environment variables, as well as some additional information. If you wish to set the trim threshold to 64KB, for example, you can run this program:

```
MALLOC_TRIM_THRESHOLD=65536 my_prog
```

Speaking of trimming, it is possible to trim the memory arena and give any unused memory back to the system by calling malloc_trim(pad). This function resizes the data segment, leaving at least pad bytes at the end of it and failing if less than one page worth of bytes can be freed. Segment size is always a multiple of one page, which is 4,096 bytes on i386. The size of the memory available to be trimmed is stored in the keepcost parameter of the struct returned by mallinfo(). Automatic trimming is done inside the free() function by calling memory_trim(), if the current value of keepcost is higher than the M_TRIM_THRESHOLD value, and by using the value of M_TOP_PAD as the argument.

Table 1. mallopt() Parameters Mapped to Environment Variables

Memory Debugging: Consistency Checks

Debugging memory is often one of the most time-consuming tasks when developing complex programs. The two basic aspects of this problem are checking memory corruption and tracing block allocation and release.

Memory corruption happens when writing to a location lying inside the legal data segment but outside the boundaries of the memory block you intended to use. An example is writing beyond an array's end. In fact, if you were to write outside the legal data segment, a segmentation fault would halt the program immediately or trigger the appropriate signal handler, allowing you to identify the misbehaving instruction. Memory corruption is thus more subtle, because it can pass unnoticed and cause a faulty behavior in a part of the program quite far from the offending part. For this reason, the sooner you detect it in the program, the higher your chances are of catching the bug.

Corruption may affect other memory blocks (messing with the application data) and the heap management structures. In the former case, the only symptom that something is going wrong comes from analyzing your own data structures. In the latter case, you can rely on some specific GNU libc consistency check mechanisms that alert you when something wrong is detected.

Memory checking in a program can be enabled as automatic or manual. The former is done by setting the environment variable MALLOC_CHECK_:

```
MALLOC_CHECK_=1 my_prog
```

This mechanism is able to catch a fair number of boundary overflows and, in some cases, to protect the program from crashing. The action undertaken when a fault is detected depends on the value of MALLOC_CHECK_: 1 prints a warning message to stderr but does not abort the program; 2 aborts the program without any output; and 3 combines the effects of 1 and 2.

Automatic checking takes place only when memory-related functions are invoked. That is, if you write beyond an array's end, it won't be noticed until the next malloc() or free() call. Also, not all the errors are caught, and the information you obtain is not always extremely useful. In the case of free(), you know which pointer was being freed when the error was detected, but that gives no hint whatsoever as to who trashed the heap. In the case of errors detected during an allocation, you merely receive a “heap corrupted” message.

The alternative is to place manual checkpoints here and there in the program. To do this, you must call the mcheck() function at the beginning of the program. This function allows you to install a custom memory fault handler that can be invoked each time heap corruption is detected. A default handler also is available if you don't provide your own. Once mcheck() has been called, all the consistency checks you get with MALLOC_CHECK_ are in place. Moreover, you can call the mprobe() function manually to force a check on a given memory pointer at any time. Values returned by mprobe() are summarized in the Sidebar “mprobe() Results”.

mprobe() Results

If you want to check the whole heap and not only one block, you can call mcheck_check_all() to walk through all the active blocks. You also can instruct the memory management routines to use mcheck_check_all(), instead of checking only the current block by initializing mcheck_pedantic() instead of mcheck(). Be aware, though, that this approach is rather time consuming.

A third way to enable memory checking is to link your program with libmcheck:

```
gcc myprog.c -o myprog -lmcheck
```

The mcheck() function is called automatically before the first memory allocation takes place—useful in those cases when some dynamic blocks are allocated before entering main().

Memory Debugging: Tracing Blocks

Tracing the history of memory blocks helps in finding problems related to memory leaks and usage or release of already freed blocks. For this purpose, the GNU C library offers a tracing facility that is enabled by calling the mtrace() function. Once this call is made, every heap operation is logged to a file whose name must be specified in the environment variable MALLOC_TRACE. Analysis of the log file then can be performed off-line using a Perl script that is provided with the library and called, not surprisingly, mtrace. Logging can be stopped by calling muntrace(), but keep in mind that applying tracing to portions of your program may invalidate the result of post-processing. For example, false leaks may be detected if you allocate one block while tracing and then free it after muntrace().

Listing 3. Tracing with mtrace()

Here is a sample tracing session using the program in Listing 3:

```
$ gcc -g Listing_3.c -o Listing_3
$ MALLOC_TRACE="trace.log" ./Listing_3
$ mtrace trace.log
Memory not freed:
-----------------
   Address     Size     Caller
0x08049718      0xa  at malloc_debug/Listing_3.c:9
```

Memory tracing has nothing to do with protection from errors; calling mtrace() won't prevent the program from crashing. Even worse, if the program segfaults, the trace file is likely to be truncated and tracing may be inconsistent. To protect against this risk, it is always a good idea to install a SIGSEGV handler that calls muntrace(), because it closes the trace file before aborting (Listing 4). More information on memory tracing can be found on the libc info page.

Listing 4. Remember to call muntrace() in the SIGSEGV handler.

Debugging Internals

Sometimes the standard debugging facilities provided by the GNU C library may not be suited to the particular needs of your program. In this case, you can resort either to an external memory debugging tool (see Resources) or carve your own inside the library. Doing this is simply a matter of writing three functions and hooking them to these predefined variables:

- __malloc_hook points to a function to be called when the user calls malloc(). You can do your own checks and accounting here, and then call the real malloc() to get the memory that was requested.
- __free_hook points to a function called instead of the standard free().
- __malloc_initialize_hook points to a function called when the memory management system is initialized. This allows you to perform some operations, say, setting the values of the previous hooks, before any memory-related operation takes place.

Hooks also are available for other memory-related calls, including realloc(), calloc() and so on. Be sure to save the previous values of the hooks and restore them before calling malloc() or free() inside your routines. If you fail to do so, infinite recursion prevents your code from working. Have a look at the example given in the libc info page for memory debugging to see all the nifty details.

As a final note, consider that these hooks also are used by the mcheck and mtrace systems. It's a good idea to be careful when using all of them combined.

Conclusions

The GNU C library offers several extensions that turn out to be quite useful when dealing with memory. If you want to fine-tune your application's memory usage or build a memory debugging solution tailored to your needs, you probably will find these tools helpful or, at least, a good starting point to develop your own mechanisms.

Resources

email: g.insolvibile@cpr.it

Gianluca Insolvibile has been a Linux enthusiast since kernel 0.99pl4. He currently deals with networking and digital video research and development.

Writing Stackable Filesystems

Erez Zadok

Issue #109, May 2003

Now you can add a feature to your favorite filesystem without rewriting it.

Writing filesystems, or any kernel code, is hard. The kernel is a complex environment to master, and small mistakes can cause severe data corruption. Filesystems, however, offer a clean data access mechanism that is transparent to user applications, which is why developers always desire to add new features to filesystems. In this article, we provide a quick introduction so you can add new functionality to existing filesystems without having to become a kernel or filesystems expert.

So You Want to Be a Filesystem Developer?

Although Linux supports many filesystems, they are pretty similar: disk-based filesystems, network-based filesystems, etc. Making a filesystem stable and efficient takes years of effort, and once it's stable and working, you don't want to break it by throwing in new features. Besides, maintainers of filesystems rarely accept feature-enhancement patches to their stable filesystems. So, it is no surprise that the most popular filesystems currently in use have not fundamentally changed in years.

Suppose you want to write a simple encryption filesystem that uses a single fixed cipher key to encrypt file data. Getting portable C code for various ciphers is easy. Next, you have to tie the calls to encrypt and decrypt data buffers into the filesystem. Conceptually the problem is simple: encrypt any data that comes from the write system call before it is written to disk, and decrypt any data that comes from the disk before it is passed back to the user process that called the read system call.

Your first inclination might be to copy the 5,000+ lines of source code for ext2, study it and then add your cipher calls to it. You should resist the urge to copy a whole other filesystem as a starting point. Although it's only 5,000+ lines of code, kernel code is at least an order of magnitude more complex to develop than user-level code. If you actually end up putting the calls to your cipher in the right place in this new filesystem, you'll find you spent most of your time studying it, only to add a small number of lines in some places. Even so, now you've got yourself a single encrypting ext2 filesystem. What if you want an encrypting NFS filesystem or any one of the plethora of other Linux filesystems?

Incremental Filesystem Development

Linux, like most OSes, separates its filesystem code into two components: native filesystems (ext2, NFS, etc.) and a general-purpose layer called the virtual filesystem (VFS). The VFS is a layer that sits between system call entry points and native filesystems. The VFS provides a uniform access mechanism to filesystems without needing to know the details of those filesystems. When filesystems are initialized in the kernel, they install a set of function pointers (methods in OO-speak) for the VFS to use. The VFS, in turn, calls these pointer functions generically, without knowing which specific filesystem the pointers represent. For example, an unlink system call gets translated into a service routine sys_unlink, which invokes the vfs_unlink VFS function, which invokes a filesystem-specific method by using its installed function pointer: ext2_unlink for ext2, nfs_unlink for NFS or the appropriate function for other filesystems. Throughout this article, we refer to the specific filesystem method using ->, as in ->unlink().

To solve this problem of how to develop our encryption filesystem quickly, we employ the following adage: “Any problem in computer science can be solved by adding another level of indirection.” Luckily, the Linux VFS allows another filesystem to be inserted right between the VFS and another filesystem. Figure 1 shows such a stackable encryption filesystem called Cryptfs. Cryptfs is called stackable because it stacks on top of another filesystem (ext2). Here, the VFS calls Cryptfs' ->write() method (cryptfs_write); Cryptfs encrypts the user data it receives and passes it down by calling the ->write() method below (ext2_write).

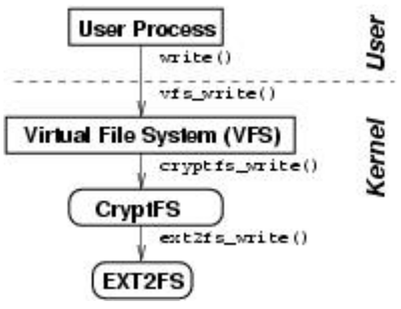


Figure 1. An Example Stackable Encryption Filesystem

In general, stackable filesystems can stand alone and be mounted on top of any other existing filesystem mountpoint; this means you only have to develop your (stackable) filesystem once, and it will work with any other native (low-level) filesystem such as ext2, NFS, etc. Moreover, as of Linux 2.4.20, stackable filesystems even can be exported safely (via nfs-utils-1.0 or newer) to remote NFS clients.

How a Stackable Filesystem Works

The basic function of a stackable filesystem is to pass an operation and its arguments to the lower-level filesystem. The following distilled code snippet shows how a stackable null-mode pass-through filesystem called Wrapfs handles the ->unlink() operation:

```
int wrapfs_unlink(struct inode *dir,
                  struct dentry *dentry)
{
  int err = 0;
  struct inode *lower_dir;
  struct dentry *lower_dentry;
  lower_dir = get_lower_inode(dir);
  lower_dentry = get_lower_dentry(dentry);
  /* pre-call code can go here */
  err = lower_dir->i_op->unlink(lower_dir,
                                lower_dentry);
  /* post-call code can go here */
  return err;
}
```

When the VFS needs to unlink a file in a Wrapfs filesystem, it calls wrapfs_unlink, passing it the inode of the directory in which the file to remove resides (dir) and the name of the entry to remove (encapsulated in dentry).

Every filesystem keeps a set of objects that belong to it, including inodes, directory entries and open files. When using stacking, multiple objects represent the same file—only at different layers. For example, our Cryptfs in Figure 1 may have to keep a directory entry (dentry) object with the clear-text version of the filename, while ext2 will keep another dentry with the ciphertext (encrypted) version of the same name. To be truly transparent to the VFS and other filesystems, stackable filesystems keep multiple objects at each level.

This is why the first few actions that wrapfs_unlink takes are to locate, from the arguments it gets, the inode and dentry that correspond to the same objects, only at the filesystem mounted below. These get_lower_* functions essentially follow pointers that previously were stored in the private fields of Wrapfs' objects when those objects were created. Once the lower objects are located, the main magic of stacking takes place. We call the lower-level filesystem's own ->unlink() method, through the lower-level directory inode, and pass it the two lower objects.

Wrapfs is a full-fledged stackable null-layer (or loopback) filesystem that simply passes all operations and objects (unmodified) between the VFS and the lower filesystem. Wrapfs itself, however, is not easy to write for one main reason; it has to treat the lower filesystem as if it were the VFS, while appearing to the real Linux VFS as a lower-level filesystem. This dual role requires careful handling of locks, reference counts, allocated memory and so on. Luckily, someone already wrote and maintains Wrapfs. Therefore, Wrapfs serves as an excellent template for you to modify and add new functionality.

Getting Started

Now that you understand how stacking works, what next? First, we have to explore the places where we can insert our code into Wrapfs. Referring back to the wrapfs_unlink code shown previously, there are three such places that correspond to before, instead of or after the call to the lower-level ->unlink() method.

1) Pre-call: you can insert code before the call to the lower ->unlink(). For example, you could check if the user is trying to delete an important file and prevent that from happening:

```
if (strcmp(dentry->d_name.name,
           "vmlinuz") == 0)
return -EACCES;
```

2) Call: you could replace the entire call itself. For example, instead of removing the file, you could rename it, as part of a simple undo filesystem (we've all had our share of unintended **rm -f** commands).

3) Post-call: here we could perform actions after the main operation returned from the lower filesystem. For example, suppose a malicious user tries to delete /etc/passwd, but the normal UNIX permission checks prevent it. An administrator might want to log such an action (using syslogd) as follows:

```
if (err == -EACCES &&
    strcmp(dentry->d_name.name,
           "passwd") == 0)
  printk("uid %d tried to delete passwd",
         current->fsuid);
```

where current is a global variable that always points to the currently executing task (process), and ->fsuid is the effective UID of that process, for use by filesystems.

These examples and those that follow have been simplified somewhat to save space and to convey their essence. For example, the d_name.name component is not null terminated, so memcmp will have to be used with the proper length; or, to check that the file referred to by the dentry is indeed the real /etc/passwd, the code has to check that the filesystem is the root filesystem or compare against the absolute pathname, using d_path(). For the complete examples, tested under 2.4.20, see the FiST home page (www.cs.sunysb.edu/~ezk/research/fist).

Example: Who Watches the Snoopers

UNIX tries to protect files from access by unauthorized users. When a user tries to open a file to which they do not have access, UNIX promptly returns a permission-denied error. Some users like to snoop around the files of others, sometimes looking for files mistakenly left unprotected or trying to guess names of files that might exist in a searchable-only directory. Unfortunately, even when those snooping users are unsuccessful, the victims of such snooping often are unaware it took place.

One of the most common filesystem operations is ->lookup(), which occurs whenever a system call uses a file's name. The kernel must translate that (string) name to an actual VFS object, such as an inode, dentry or file. To detect snooping users, we place the following code in snoopfs_lookup or snoopfs_permission, right after it calls ->lookup() on the lower filesystem:

```
if ((err == -EACCES ||
     err == -ENOENT) &&
    dir->i_uid != current->fsuid &&
    current->fsuid != 0)
  printk("snoop uid=%d pid=%d file=%s",
         current->fsuid, current->pid,
         dentry->d_name.name);
```

Here, we check the return code (err) from the call to the lower ->lookup(). If the status is EACCES (permission denied) or ENOENT (no such file or directory) and if the directory's owner (dir->i_uid) is different from that of the user running the current task (current->fsuid) and the current user is not the superuser (because root users can do anything), then it prints a descriptive message identifying the snooping user. This message typically is logged by syslogd.

Example: Encryption Fit for a Caesar

Wrapper technologies are particularly suitable for security-related applications where wrapping, or monitoring, is often useful. Not surprisingly, the most popular applications developed from FiST are cryptographic filesystems. In this example, we demonstrate a simple encryption filesystem that uses the rot13 cipher.

In this filesystem, we want to encrypt all file data using a function (presumably already written) called rot13 that takes an input buffer, output buffer and their lengths. However, unlike the previous examples, there is no single method where you can place the rot13() function to encrypt the file's data. In fact, manipulating file data in any filesystem is rather complex because it involves multiple methods, as well as two forms of accessing file's data, the read and write system calls, which can work at any file offset, and mmap, which works on whole pages. To make life easier for stackable filesystem developers, Wrapfs consolidates all of these methods into two simple calls: one to encode file data and one to decode file data, both working on whole page-aligned data pages (for example, 4KB on IA-32 systems). Using the Wrapfs template, the only code you have to write to produce a rot13-based encryption filesystem looks like the following:

```
int
encode_block(void *in, void *out, int len)
{
  rot13(in, out, len);
  return len;
}
int
decode_block(void *in, void *out, int len)
{
  rot13(in, out, len);
  return len;
}
```

Wrapfs already contains all the complex code that handles mixed reads, writes and memory-mapped operations. Wrapfs makes calls to encode_block to encrypt a data page and to decode_block to decrypt a data page (they are identical in this example).

Of course, rot13 is hardly a practical cipher, but given this simple example, you can build much stronger cryptographic filesystems. Following this, we recently have built a powerful cryptographic filesystem called NCryptfs (a successor to Cryptfs). NCryptfs supports multiple ciphers; multiple keys per user, process or group; multiple authentication schemes; key timeouts and revocation; delegated privileges; and more—all with a negligible performance overhead.

Wrapfs also supports manipulating filenames using two additional routines to encode and decode filenames. One thing to watch out for when encrypting filenames is that filenames must remain valid after encryption. In other words, they cannot contain nulls or “/” characters. A common solution is to uuencode the file's name after encryption.

Example: Extending New Functionality to User Applications

In the wrapfs_unlink example, we suggested that instead of deleting a file, you could rename it, thus saving a single backup of deleted files. Suppose we call this filesystem unrmfs, in which deleted files are instead renamed from their original name *F* to *F*.unrm. It might be annoying if all of these .unrm files started appearing in your directory, especially if you're expecting nothing there. Moreover, this kind of functionality also could be used to fool attackers who try to delete log files that may be used to track their actions. To achieve this, however, the .unrm files must not be visible or accessible to users by default.

To hide certain files in a filesystem, you have to do two things. First, prevent the file from showing up in ->readdir(). This is done by writing code in wrapfs_filldir that checks each filename passed to ->filldir() and returning NULL for those files you do not want listed. Second, prevent users from directly looking up the file by its name; this is done by checking for .unrm files in the beginning of wrapfs_lookup.

Of course, hiding those files from all users isn't very useful. Legitimate users must be able to access those files under certain conditions. A simple approach might be to check the UID of the calling process and to hide the .unrm files only from certain users. A better approach would be to use the mother of all system calls, ioctl. In Wrapfs, you can define as many new ioctls as you like, and then write small user-level programs to use those ioctls. This is, for example, the mechanism we use in encryption filesystems for a user-level tool to pass a user's cipher key to the kernel.

For our unrmfs, you could write a restore ioctl that takes a file's name, *F*, checks whether the file *F*.unrm exists and then renames *F*.unrm back to *F*, effectively unhiding it from unrmfs. The following example shows a sketch of this code:

```
/* len: length of source file */
newname = kmalloc(len+6, GFP_KERNEL);
strncpy_from_user(newname, ioctl_arg, len);
strcat(newname, ".unrm");
lower_dir = get_lower_inode(dir);
src = lookup_one_len(lower_dir, newname);
if (IS_ERR(src))
  return PTR_ERR(src);
dst = lookup_one_len(lower_dir, name);
vfs_rename(lower_dir, src, lower_dir, dst);
```

Conclusion

Filesystem development need not be difficult. Using stackable filesystems, you can create new, useful and efficient filesystems quickly—all without changing kernels or existing filesystems. The examples in this article hopefully demonstrate the power of stacking, from which you gradually can build more complex filesystems. You can get the FiST software, documentation and many more examples from www.cs.sunysb.edu/~ezk/research/fist. Happy stacking.

email: ezk@cs.sunysb.edu

Erez Zadok (ezk@cs.stonybrook.edu) is on the Computer Science faculty at Stony Brook University, the author of Linux NFS and Automounter Administration (Sybex, 2001), the creator and maintainer of the FiST stackable templates system and the primary maintainer of the Am-utils (aka, Amd)

automounter. Erez conducts operating systems research with a focus on filesystems, security and networking.

Archive Index Issue Table of Contents

Advanced search

Introducing Plone

Reuven M. Lerner

Issue #109, May 2003

Is Plone the killer app that will bring Zope-based content management to the masses?

One of the biggest reasons for the success of the Apple II computer in the late 1970s was not a product of Apple Computer, Inc. Rather, the first spreadsheet, VisiCalc, written by Dan Bricklin and Bob Frankston, helped propel sales of the Apple II. VisiCalc was a revolutionary product, and it worked only on Apple computers. Businesses everywhere began to buy Apple computers, simply so they could have VisiCalc.

It turns out that Bricklin and Frankston hadn't invented only the spreadsheet. They also had invented the killer app—an application that makes a platform so compelling, people adopt the platform for that one product. Every platform vendor hopes someone will produce a killer app for its product, driving up sales of the platform technology as more and more people want it for themselves. Things are not much different in the Open Source world, although different motivators exist. For example, many people have adopted Linux in order to use Apache or Python to work with Zope.

The latest example of a killer app in the Open Source world is Plone, a content management system (CMS) written on top of Zope by Alexander Limi, Alan Runyan and Vidar Andersen. Plone has become an increasingly prominent piece of software in recent months and has brought many people into the Zope community. And although many Plone users seem to stay within that world and don't venture into the depths of the Zope server, Plone may well be the killer app for Zope. If true, this would be a fascinating trend, given how Zope has long been considered the killer app for the Python language.

What Is Plone?

Plone is a simple CMS that allows a site administrator to grant different privileges to different users. All users can read publicly available content. Some users are allowed to enter new content, and other users are allowed to publish that content to the general public. Indeed, the distinction between *available* and *published* content is what most distinguishes a CMS from a run-of-the-mill web site.

In contrast with a static web site, where files are available to the public as soon as they exist in the root document directory, a CMS allows you to expose content selectively. Moreover, a CMS allows you to retract previously published content. So if your site publishes a news story that turns out to be false, you can remove it from the public's view without actually having to remove any files. A log of when the story was written, published and retracted, along with who performed each action and the reason why, is available to site administrators at all times.

Plone is not designed to be the be-all and end-all of content management systems. Rather, it is meant to be used on small- and medium-sized web sites whose administrators want to provide a variety of useful features, prefer a nice user interface and need the ability to customize the site's look and feel to a certain degree.

Plone itself is implemented as a number of different Zope products, where each product is actually an object class that can be instantiated multiple times. However, Plone is not implemented directly in Zope but within Zope's content management framework (CMF), a set of objects and APIs meant to make it easy to create your own CMS.

Plone 1.0 was released prior to this writing, in early 2003, and depends on CMF v1.3, which was released in mid-2002. Just as a desktop application uses many of the facilities that the underlying operating system provides, Plone, or any CMF-based CMS, uses the capabilities that the CMF provides. Plone sites thus offer full-text search and the ability for community members to comment on any content object. As the CMF improves and offers more services, I expect Plone also will improve.

Installing Plone

If you already are running Zope, installing Plone is quite easy. Remember that every Zope product must be installed in the lib/python/Products directory within your Zope directory. In addition, Zope must be restarted either manually or from the web-based control panel in order for newly installed products to be seen and registered.

Before you can install Plone, you must install the latest version of the CMF. Retrieve CMF-1.3 from cmf.zope.org, which comes as a gzipped tar file. I put the tar file in /tmp and installed it as follows:

```
# cd $ZOPE/lib/python/Products
# tar -zxvf /tmp/CMF-1.3.tar.gz
```

The CMF-1.3 directory created in lib/python/Products contains a number of CMF-related products that Zope needs to locate at startup. We therefore create a number of symbolic links to the product directories:

```
# ln CMF-1.3/CMFCore .
# ln CMF-1.3/CMFCalendar .
# ln CMF-1.3/CMFDefault .
# ln CMF-1.3/CMFTopic .
```

Now that the CMF is installed, we can install Plone as well. Retrieve a tar file of Plone from www.plone.org, place it in /tmp and expand it:

```
# cd $ZOPE/lib/python/Products
# tar -zxvf /tmp/CMFPlone-1.0.tar.gz
```

As with the core CMF product, you must create several symbolic links in the Products directory, so Zope can recognize them when it starts up:

```
# ln -s CMFPlone-1.0/CMFPlone .
# ln -s CMFPlone-1.0/DCWorkflow .
# ln -s CMFPlone-1.0/Formulator .
```

If you are using a version of Zope prior to 2.6.x, you might need to create a symbolic link from another product to the main Products directory. Check the Plone instructions to be sure.

Make sure the CMF-1.3 and CMFPlone directories and their contents are owned by the same user as the one as which Zope runs. This is normally a user named www or zope. Running Zope as nobody, which used to be considered a safe option, is no longer recommended. If the appropriate user does not own the files, you could encounter some odd ownership and permission problems.

Now you have installed everything you need to create a site managed by Plone. Start Zope, log in as a user with administrative privileges and you're ready to go.

Creating a Plone Site

Creating a Plone site is extremely easy from within the web-based Zope management interface. From the add product menu, choose the Plone site. You will be prompted to enter several pieces of information:

- the ID of the site, which will be part of the URL;

- the title of the site, which will appear at the top of each page;
- whether the site should have its own user folder or should inherit users from the surrounding Zope site;
- a description of the site; and
- the type of site you're creating (for now, leave it at the default Plone).

When Zope finishes creating a new instance of the Plone site, the large pane in the Zope management interface changes dramatically. An introductory message appears in the middle, a toolbar slides across the top, and information, including a calendar, shows up in rectangles along the left and right sides.

The interface for modifying a Plone site is significantly different from the standard Zope interface. Whereas Zope normally displays the same screens for all users and only presents the management interface when the /manage method call is tacked onto a URL, Plone modifies its output according to the current user's permissions. So although guests can navigate only through the site's content, administrators are shown tabs, such as view, edit, properties and state, and can see all items on the site, including those not yet published.

Luckily, the Plone user interface is fairly straightforward for administrators. To modify the contents of a page, click on the edit tab. You can then edit the content, including its URL and summary information, using your web browser. The summary information appears in search results. Plone uses JavaScript to make the user interface easier to understand for nontechnical users; for example, clicking on any HTML widget, such as a text area or radio button, in the Plone editing interface brings up a tooltip-like description of what should be inserted.

Content added or modified using the web-based Plone interface can be in plain text, in HTML or in Zope's structured text format, which uses punctuation and indentation as formatting mechanisms. I prefer to use structured text as often as possible, using HTML only when I want to format a page in a way that structured text doesn't allow.

Surrounding the main document on a page are multiple portlets, accessories that add to a site's content. Plone comes with several portlets by default, including a list of news stories, a list of events, a calendar that displays today's date and highlights any events during the current month and a list of relevant documents on the current site.

To add a new document, first move to the contents view by clicking on that link in the navigation portlet. This produces a list of documents in the current folder. You then can select a new content type from the add new item in the upper-left corner. It's important to realize a new document is created as soon as you click on add new document; following that, you are modifying its properties and content.

By default, Plone allows you to create a number of different content types:

- Folders allow you to structure your site with a hierarchy. Just as a disk, static web site or Zope site contains files in folders, Plone sites can contain folders. The title of each published folder is displayed in the navigation portlet.
- Documents are the most common item on Plone sites and can be formatted in HTML, structured text or plain text. Most of the time, you probably will want to create a new document.
- Images can be in nearly any format, including JPEG, PNG and GIF.
- Files are items you want users to be able to view or download but that don't have a MIME content type that Plone can work with easily. Some examples are QuickTime movies, audio clips and Microsoft Office documents.
- Events are short documents with a starting and ending date and are highlighted in the calendar portlet.
- News items are short documents displayed in the news portlet. This is a good and easy way to publish press releases, for example.

Plone also comes with links, which are URLs of interest to the outside world, and topics, which are predefined searches within the site. An increasing number of other content types are emerging for Plone, such as a weblog and a photo album.

Publishing Documents

When you create a new piece of content in Plone, it is in the visible state by default. This means that if people know the URL, they can get to your document with their web browsers. The content does not appear, however, in searches or in the navigation portlet.

To publish content, click on the state tab at the top of the page. (From the contents view, you can publish multiple pieces of content simultaneously with the state button at the bottom of the page.) This brings you to a page that asks when the content should first be published, when it should expire and any comments you wish to make about the decision to publish the document. The dates you enter are the final arbiter, meaning that a published document will be visible only between its starting and ending dates. This allows you to enter content days or weeks before it should be exposed to the general public, without having to change its state to published at the appropriate time.

Once published, a document appears in full-text searches. It also is visible within any folder that lacks a default index_html document.

One of my favorite Plone features is properties. Each document can be assigned one or more properties by the properties management tab at the top of the screen. When a user views the content, the related portlet lists all of the other documents on the site that share one or more properties with the current one. This allows site visitors to find easily other content that might interest them.

Conclusion

Zope is a powerful application server, and the CMF is a powerful toolkit for creating a CMS. But, given the steep learning curve associated with both of them, it is possible that Plone may be the killer app for Zope, bringing new people into the world of Zope through great functionality that is easy to install, configure and manage.

Next month, we will take a closer look at Plone, examining ways we can modify what is displayed, including changing the site's look and feel.

Resources

Reuven M. Lerner (reuven@lerner.co.il) is a consultant specializing in open-source web/database technologies. He and his wife, Shira, recently celebrated the birth of their second daughter, Shikma Bruria. Reuven's book Core Perl was published by Prentice Hall in early 2002, and a second book about open-source web technologies will be published by Apress in 2003.

Battles inside the Computer

Marcel Gagné

Issue #109, May 2003

You won't look like Jeff Bridges, but do you get to wear the funky costumes?

What do you mean, “Who is inside the computer?” No one is inside the computer, François. Ah, I see. I think you misunderstand the concept of this month's focus, “Kernel Internals”. Why, the whole notion of there being anything other than bits and bytes floating in the Linux kernel is pure science fiction, *mon ami*. *Qu'est-ce que tu dis*? No, of course not. Inside the computer isn't a real place, at least not in this world. Still...

Quoi? Sorry, *mon ami*. I was just remembering a movie from years ago, where there really was an “inside” the computer, and programs fought for their users against the MCP, the master control program. François, what are you looking at?

Ah, *mes amis*. Welcome to *Chez Marcel*, home of fine Linux fare and excellent wine. Please sit and I will have François fetch the wine. We have some incredible French Canadian *tourtière* to serve tonight as well. François, run to the cellar and fetch the wine. The 1997 Napa Valley Cabernet Sauvignon is drinking exceptionally well.

It is good to see you, *mes amis*. François had this strange notion that kernel internals implied something was going on inside his Linux system that involved *real* people. That made me think of the 1982 Disney film called *Tron*. Inside the computer, programs are engaged in gladiatorial battles, fighting for their users. One of the deadly sports in this virtual world was a lightcycle race. Contestants rode a kind of motorcycle that left a wall of light in its wake. The cycles themselves can't stop. The only thing you can do is ride, avoiding your opponents' walls while trying to get them to crash into yours. The last program standing wins.

François, you have returned. Please pour for our guests, then head back to the kitchen for that *tourtière*. As I was saying, *mes amis*, as dated as *Tron* appears in today's world, the lightcycle battle was the inspiration for various arcade games based on the simple concept of an ever-growing light trail that must be avoided at all costs. That inspiration continues today and is represented in several open-source projects.

Let's start with a lightcycle game you already may have on your system. As part of the KDE games package, you'll find Matthias Kiefer's *KTron*, a simple but highly functional lightcycle game. You'll find it in your K menu under arcade games, and the command name is **ktron**. When the game starts, you'll see two squares near the center of the screen, one red and one blue. To play, press either your left or right cursor and the race begins instantly (Figure 1).

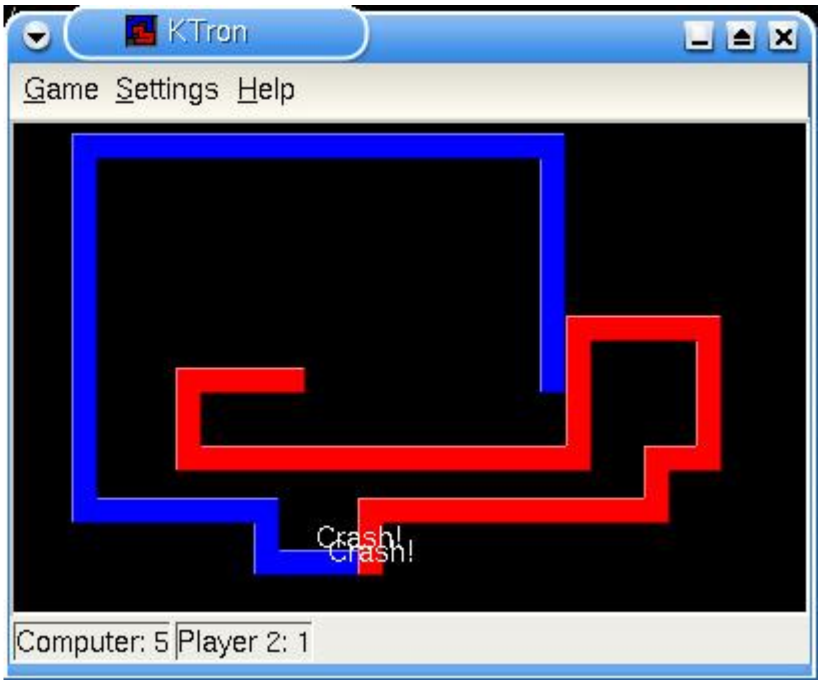


Figure 1. *KTron*: KDE's Own Lightcycle Game

In order to change game play, click Settings on the menubar and several modifications are at your disposal. For instance, if the game is a little fast for you, tone down the velocity. Or, speed it up if you are falling asleep at the lightcycle's wheel. You also can change the size of the cycle's trail or modify the default colors.

This little game is actually a lot of fun, particularly if you change the size of the arena, but really to get the feel of a lightcycle battle, you must enter the three-dimensional world. That, *mes amis*, is how you get into the computer. You will need a 3-D accelerated video card and OpenGL or Mesa 3-D video libraries.

As far as the card itself is concerned, many are extremely well supported under Linux. Unfortunately, even when the card is supported, the manufacturer of the card may not license the accelerated driver for inclusion with the Linux distributions themselves. That doesn't mean they aren't available, but you may have to visit the vendor's site and download them. With some cards, full 3-D acceleration is directly supported under XFree86. So, how can you tell if you are ready to go? A quick way to test for the presence of 3-D support is with the following command:

```
glxinfo | grep rendering
```

The system should respond with this:

```
direct rendering: Yes
```

Another fun little test you can perform involves running a program called **gears** (part of the Mesa-demos package). To do this, simply type the command **gears** at a shell prompt. Your hard work will be rewarded with three gears spinning on your screen (Figure 2).

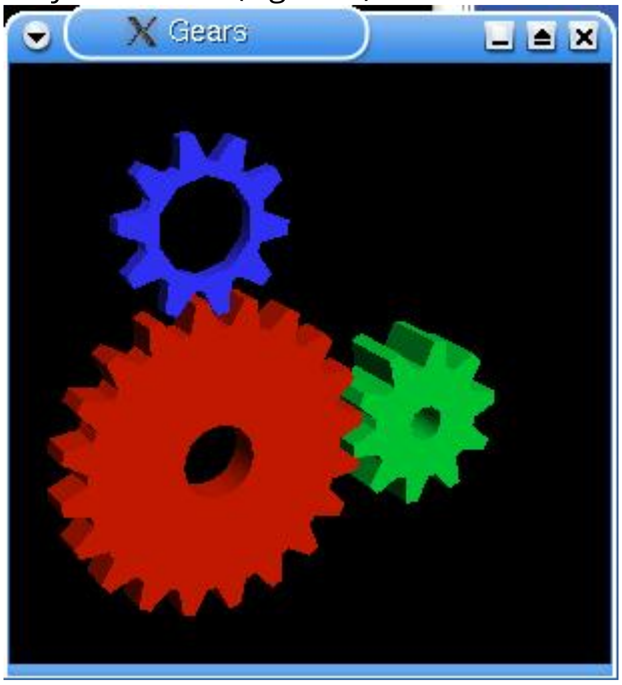


Figure 2. The 3-D gears Demo from Mesa

Do not get too distracted by the spinning gears. Have another sip of your wine, then look back at your terminal window. You will see some statistics regarding the performance of your 3-D hardware:

```
1778 frames in 5.001 seconds = 355.529 FPS
```

That result comes from my test system, which has an NVIDIA GeForce2 card. This is actually a reasonably impressive hardware performance. In contrast, my little notebook, which suffers from a complete lack of accelerated hardware, yields this result:

```
  312 frames in 5.004 seconds = 62.350 FPS
```

Not having acceleration for your card doesn't stop the program from working; it simply will run very, very slowly. Now that we've got that out of the way, *mes amis*, let us return to the world of lightcycles in the third dimension with Manuel Moos' *Armagetron* (armagetron.sourceforge.net). As Manuel's home page suggests, think Armageddon with a “tr” instead of that double “d”. Precompiled binaries were easy to find, and the source, of course, is there as well.

The first time I tried the game, I immediately crashed into the wall. This was my experience the first three times or so, until I realized that perhaps I needed to configure my keys for steering. Navigate the menus to Player settings (whichever player number you happen to be), then change the Input Configuration. Cursor down to turn left and press Enter. You'll be asked to press the key you wish to use for a left turn. I know it sounds crazy, but I chose the left cursor key. Repeat the process for the right turn and anything else you may wish to use in combat, such as instant chat phrases so you can taunt other players with a single keystroke. Exit that menu, then the one before it until you return to the main menu.

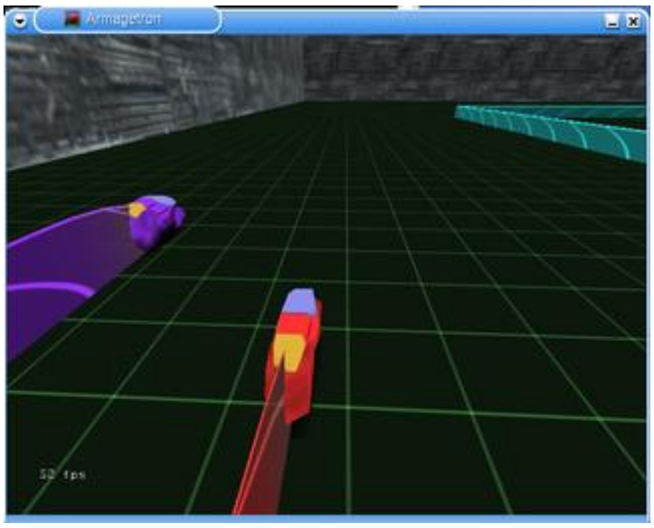


Figure 3. A 3-D Lightcycle Race, Courtesy of *Armagetron*

Now, from the Game menu, select Start Game. *Armagetron* is a great lightcycle game with an omnipresent overhead camera that follows your moves and provides some dizzying perspective changes on those instant, 90-degree turns (Figure 3). *Armagetron* also is network playable with up to 16 computers, each running four players.

Armagetron is wonderfully addictive and, as a result, comes with an absolutely essential feature: a Boss key. In other words, if you happen to be playing a game of *Armagetron* over the network and the boss approaches, press Shift-Esc, and the game ends and vanishes instantly.

Armagetron is great, but for something approaching photo-realism, we have Andreas Umbach's *GLTron*. As you might expect, *mes amis*, this realism comes at a price in terms of hardware and performance, but if you have the machine, you will not be disappointed. You should have no problem finding precompiled binaries (RPMs are available on rpmfind.net), but the source is always available. You can pick it up at www.gltron.org.

GLTron is fast-paced (assuming proper 3-D hardware) with impressive graphics and a play mode that is sure to raise your heart rate.

There are three main configuration menus for *GLTron* (game, video and audio), and some of these make it possible to change the game experience dramatically. For instance, you can run *GLTron* in full-screen mode as well as windowed. Artpacks also are included in the package so you can alter the look and feel. My favorite is still the default (Figure 4), but I am somewhat fond of the Metaltron artpack.

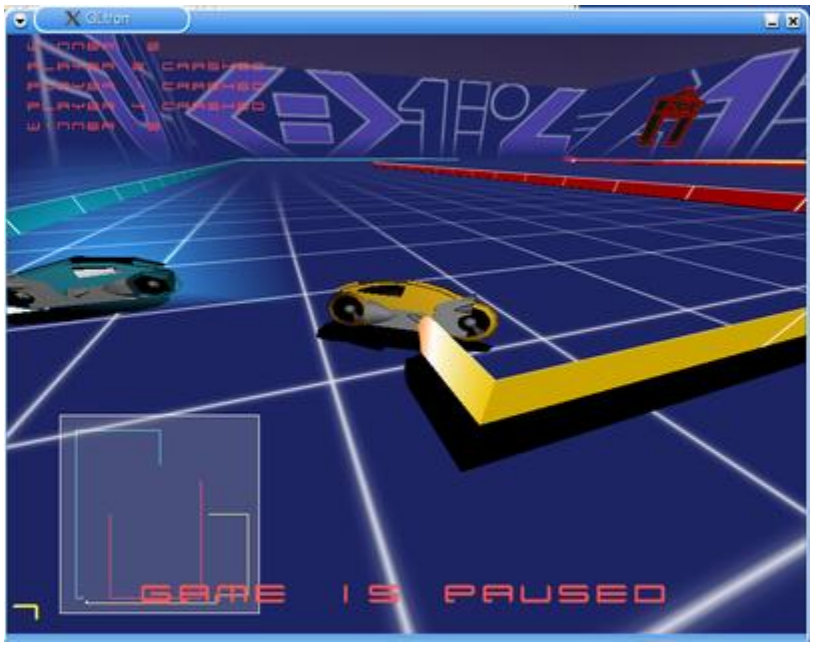


Figure 4. *GLTron*--So Good, You'll Feel You're in the Computer

What I like most about *GLTron* is the ability to pan with the mouse to create almost any camera view. Press the spacebar to pause the game, then tilt or rotate your point of view. If you really want to hone your skills in this 3-D world, try changing your point of view relative to the arena and restart the race. Controlling your lightcycle from the side is an altogether different experience than it is from above or behind. Try pressing F10 to change the camera angle and tracking. I find the behind-camera viewpoint particularly interesting. It changes your view so that you seem to be riding in the lightcycle itself—very heady.

By default, a small 2-D map floats above the screen. If you feel particularly cocky, try turning the map off.

In many ways, *GLTron* is the most polished of the three games mentioned here, but it doesn't have the network play of *Armagetron*. On the other hand, as simple as *KTron* is, you can play it on any system—accelerated hardware is not necessary.

So, *mes amis*, as you can see, all your workstations are set up with the three games as well as being fully networked. Perhaps you would like another glass of wine before entering the arena? Or maybe an espresso? François will be happy to provide you with either.

Mon Dieu, has the time passed so quickly? As I cannot connect to your systems and pour you a glass of wine, I must attend to it here before François and I close the restaurant for the night. François, would you be so kind as to refill our guests' glasses a final time? Until next time, *mes amis*, let us all drink to one another's health. *A votre santé*! *Bon appétit*!

Resources

Marcel Gagné lives in Mississauga, Ontario. He is the author of Linux System Administration: A User's Guide (ISBN 0-201-71934-7), published by Addison-Wesley (and is currently at work on his next book). He can be reached via e-mail at mggagne@salmar.com.

Using Firewall Builder, Part I

Mick Bauer

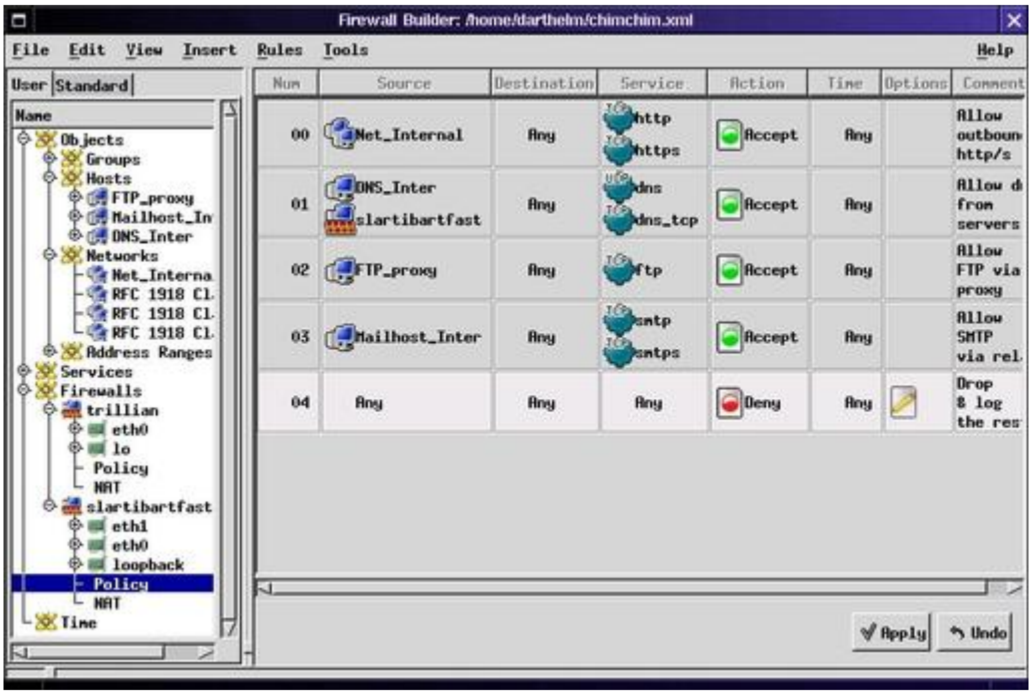
Issue #109, May 2003

Use one easy GUI application to build and deploy policies for all your firewalls and servers.

Linux 2.4's Netfilter firewall code and its front end, iptables, deserve the praise and popularity they've garnered. They've brought Linux firewalls to the same level as commercial stateful packet-filtering firewalls, from the standpoints of functionality, intelligence and security.

Only one thing has been lacking from the Netfilter experience: user-friendliness. A good firewall GUI isn't merely a crutch to be used by nontechnical people. Even the most pointy-headed of us tend to work faster and make fewer mistakes in our firewall policies if we can construct rules with the aid of visual cues and reminders. There's little value in focusing on iptables' command syntax at the expense of the actual security policy your firewall needs to enforce.

Firewall Builder (Figure 1) is a good firewall GUI indeed. It lets you define host, network and service objects that can be used and reused in as many different firewall rulesets as you like; it displays your rules in an instinctive and clear way; and because it's intentionally OS-agnostic, you can use Firewall Builder to generate rulesets not only for Netfilter/iptables, but also for FreeBSD's ipfilter, OpenBSD's pf and even Cisco PIX firewalls.



Num	Source	Destination	Service	Action	Time	Options	Comment
00	Net_Internal	Any	http https	Accept	Any		Allow outboun http/s
01	DNS_Inter slartibartfast	Any	dns dns_tcp	Accept	Any		Allow d from servers
02	FTP_proxy	Any	ftp	Accept	Any		Allow FTP via proxy
03	Mailhost_Inter	Any	smtp smtps	Accept	Any		Allow SMTP via rel
04	Any	Any	Any	Deny	Any		Drop & log the res

Figure 1. Firewall Builder in Action

This issue and next I'll show you how to obtain and install Firewall Builder, and then I'll explain how to use it to build iptables policies of your own easily and instinctively. To begin, we focus on installing Firewall Builder and on populating its Objects database; next month we'll cover policy construction in-depth.

Where to Install Firewall Builder

First, a few words on where to install and run Firewall Builder. I don't think it's a good idea to run Firewall Builder on an actual firewall or on any other hardened, publicly accessible host, called a bastion host. In short, I don't think you should run the X Window System on such hosts.

Instead, you should run Firewall Builder on your normal day-to-day workstation. Then, copy the firewall scripts you build to the host you actually wish to configure, using scp or some other secure means. Firewall Builder is designed to be used in this way.

On the other hand, if you intend to use Firewall Builder to create Netfilter scripts for local protection of one particular host, such as a Linux 2.4-based web server, perhaps it's okay to run Firewall Builder directly on the host on which its scripts will be installed. But, make sure X11 is installed on the host and the host itself is behind a proper firewall.

The important point is you don't need to run Firewall Builder on each host you want configured. Therefore, you shouldn't run it on any host on which you wouldn't otherwise run the X Window System. A single host running Firewall

Builder can generate scripts for as many different hosts as you like. We'll see how this is possible shortly.

Getting and Installing Firewall Builder

Naturally, the Firewall Builder Project has its own home page, where you can obtain the latest software releases and documents: www.fwbuilder.org. If anything I say here is different from something you read there, I defer to that site. Firewall Builder's on-line installation instructions are clear and accurate, and they may change between the time I wrote this article and the time it actually is printed.

Debian

I'll start with the easiest case. If you run Debian 3.0, you can install Firewall Builder directly from your Debian installation source; Debian has its own officially supported deb package, called fwbuilder. Among other things, this package depends on the Debian packages libfwbuilder0, fwbuilder-iptables, libgtk1.2, libgtkmm1.2, libxslt1, libxml2 and libsnmp4.2.

I'll skip the complete list of dependencies, though. If you use apt-get to install fwbuilder, apt-get will identify and install all required packages for you. I also recommend installing Debian's fwbuilder-doc package; it is optional (and therefore won't be installed automatically by apt-get in order to satisfy any dependencies) but contains comprehensive and useful documentation.

Red Hat

As of Red Hat 8.0 (the latest Red Hat release at the time of this writing), Firewall Builder isn't yet an official part of Red Hat Linux. However, the Firewall Builder team provides RPM files for several Red Hat releases; see the Firewall Builder download site at sourceforge.net/project/showfiles.php?group_id=5314.

You'll need the fwbuilder and libfwbuilder packages, plus at least one of fwbuilder-ipt, fwbuilder-ipf or fwbuilder-pf, depending on whether you create rulesets for Linux Netfilter/iptables, FreeBSD ipfilter or OpenBSD pf, respectively. You can install more than one of these last three if you wish. Because Firewall Builder's ultimate output is an ASCII script, using a Linux system to generate firewall rules for other platforms is not a problem.

Before installing the Firewall Builder packages, the following standard Red Hat packages must be present: bind-utils, gdk-pixbuf, glib, glibc, gtk+, gtkmm, libfwbuilder, libsigc++, libstdc++, libxml2, libxslt, openssl-0.9.6b, ucd-snmp and XFree86-libs.

In addition, you'll need gtkmm (the GIMP Tool Kit Minus Minus), which contains the C++ bindings for GTK+. This package is part of Ximian GNOME, but you also can download it from www.freshrpms.net.

SuSE

Like Red Hat, SuSE has not yet incorporated Firewall Builder into its official release. SuSE 8.1 RPMs (albeit unofficial contributed ones) are available from the Firewall Builder download page (sourceforge.net/project/showfiles.php?group_id=5314).

You'll need the fwbuilder and libfwbuilder packages, plus one or more of fwbuilder-ipt, fwbuilder-ipf or fwbuilder-pf. You'll also need to have installed these standard SuSE packages: gcc, gdk_pixbuf, glib, glibc-2.2.4, gtk, gtkmm, libsigc++, libstdc++, libxml2, libxslt, libz, openssl-0.9.6b, ucdsnmp and xshared.

Creating Objects

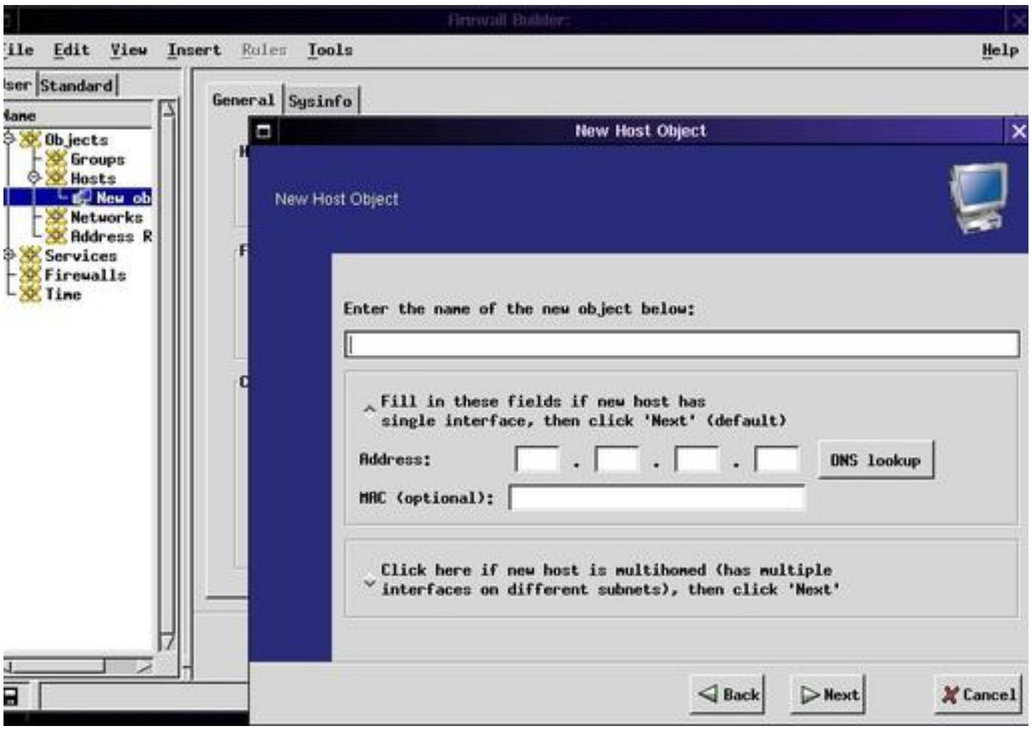
Once its packages are installed, you're ready to run Firewall Builder. There's only one command to remember: **fwbuilder**. You must have the X Window System running to run this command. You don't need to be root, though; in fact, I recommend against it, because you always should avoid needlessly doing things as root.

Once the fwbuilder window is up, you can start defining objects (Figure 2). The whole point of Firewall Builder is to be able to write rules using reusable, drag-and-drop objects, so obviously, objects must exist before rules may be written. Even Firewall Builder's automatic-policy-generating druids require that objects already exist.

You can use objects to represent hosts, networks (identified by IP address and subnet mask), address ranges, TCP/IP services, firewalls (both multi-homed firewalls proper and bastion hosts), time-ranges and groups of other objects. You may define as many or as few objects as you think you'll use in your rulesets. At a minimum, you'll need a firewall object and at least one network or host object. Predefined service objects are provided for many popular TCP/IP services.

Host Objects

Objects are created from the dialogs in Firewall Builder's Insert menu. Figure 2 shows the Insert host dialog for creating a host object. A host object's defining characteristic for rule-creating purposes is its IP address. If you wish to write rules that match hosts by MAC/Ethernet address, you can define that too. As you can see, you may enter the IP address manually or by DNS lookup. The latter feature is handy, but it works only for hosts whose names are resolvable by the system on which Firewall Builder is running.



Enter the name of the new object below:

Fill in these fields if new host has single interface, then click 'Next' (default)

Address: . . .

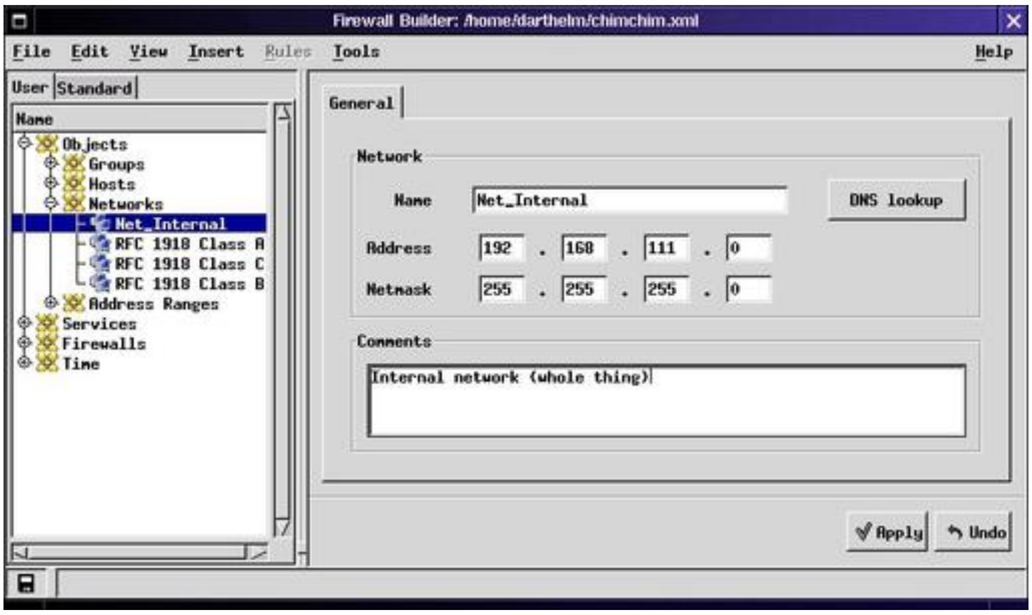
MAC (optional):

Click here if new host is multihomed (has multiple interfaces on different subnets), then click 'Next'

Figure 2. Insert Host Dialog

Network Objects

Figure 3 shows the Insert network dialog. Unlike Insert host, which pops up a separate window, Insert network simply opens a blank New object form in the right-hand portion of the main window. This dialog actually is simpler than the Insert host dialog; all you need to enter are the network's IP address and subnet mask, a name for the network object and, optionally, a comment.



Network

Name: Net_Internal

Address: 192 . 168 . 111 . 0

Netmask: 255 . 255 . 255 . 0

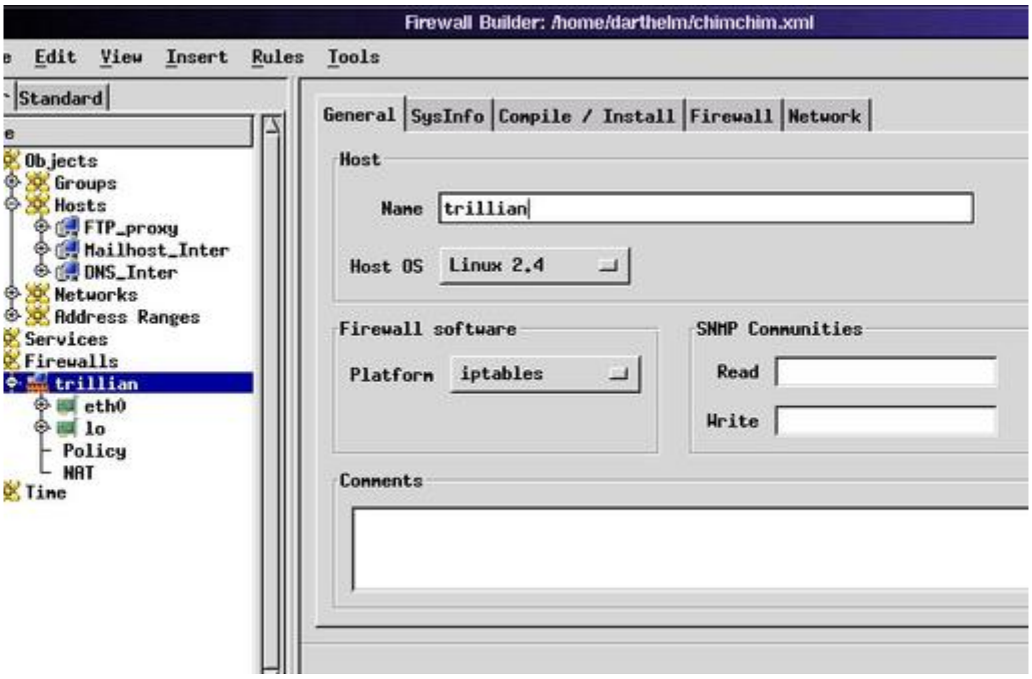
Comments

Internal network (whole thing)

Figure 3. Insert Network Dialog

Firewall Objects

The most complicated object by far is the firewall object. Initial setup isn't too hairy in itself; simply define the firewall's interface or interfaces by IP address and subnet mask. But once the firewall object has been added, and it appears in the list of user-defined objects on the left-hand side of the main fwbuilder window, click on its icon. Five pages (tabs) of information should appear on the right-hand side of the window (Figure 4).



Name: trillian
Host OS: Linux 2.4
Platform: iptables

Figure 4. Firewall Properties

On the General screen, we see the hostname entered in when the firewall object was created. It's important, though, to select appropriate Host OS and Platform options, so Firewall Builder will know which of its compiler engines to use when converting policies to firewall scripts for this firewall.

The SysInfo screen applies only to SNMP data (see Sidebar). The Compile/Install screen is where you can set up automatic installation of your firewall policies. If you intend to install them manually, you can leave this screen alone. Someday, hopefully soon, Firewall Builder will support the automatic transfer and installation of firewall scripts using SSL. As of this writing, however, the fwbd dæmon that must run on a target firewall for this method to work has not been released.

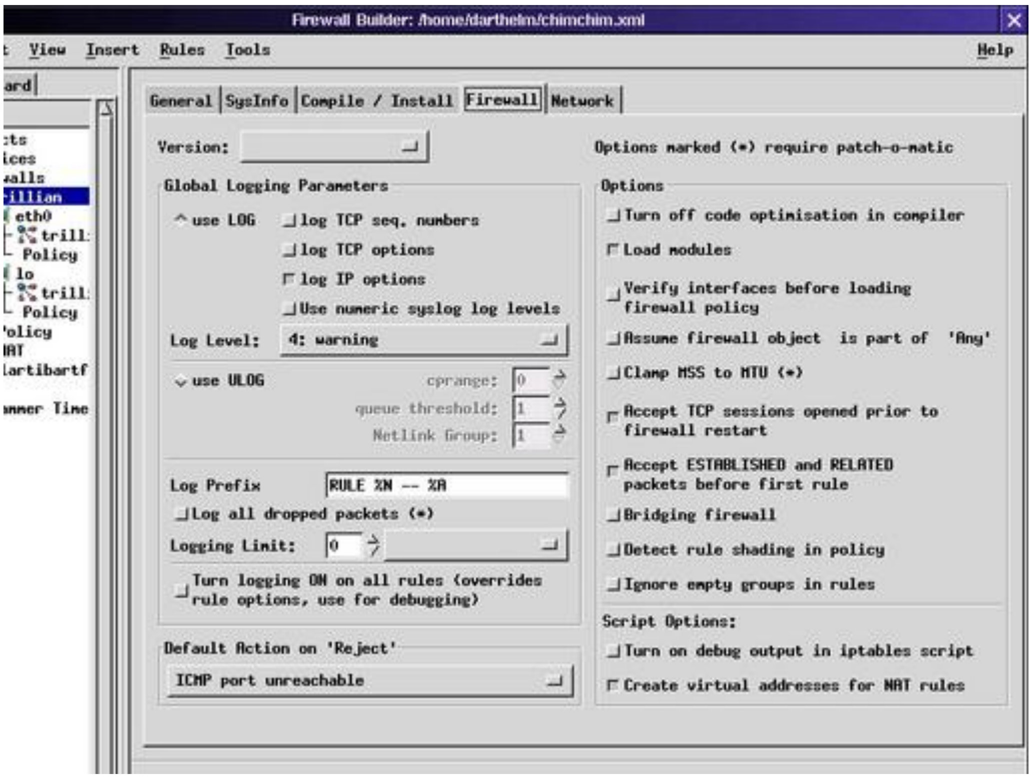
If you leave the Compile/Install screen's Installer option at its default of fwbd, even though this feature isn't yet supported, nothing bad will happen; compiled firewall rules still are saved to your home directory. The Install option in the Rules menu will be grayed out, though. If, on the other hand, you set Installer to Install Script, you then can specify the path to a custom script in the Policy

Install Script field, with optional command-line parameters below. The custom script will be executed when you select Rules®Install after compiling a policy.

This method is a handy way to script, for example, an scp command that securely copies your policies to their target firewalls. Sample installation scripts, notably fwb_install, are available under contrib at the Firewall Builder download site (sourceforge.net/project/showfiles.php?group_id=5314).

Regardless of what you set Installer to, Firewall Builder writes the script it compiles for this firewall to a local ASCII file with the same name as the firewall object and a suffix of .fw. For example, scripts generated for the firewall Trillian in Figure 4 are named trillian.fw.

Continuing with Firewall's object properties, the Firewall tab is used to configure options specific to the platform you selected in the General screen, in our case Netfilter/iptables-specific options (Figure 5). The defaults here work fine for many if not most users, but a couple of the options are worth discussing.


Firewall Builder: /home/darthelm/chimchim.xml
t View Insert Rules Tools Help
ard
ts
ices
alls
illian
eth0
trill
Policy
lo
trill
Policy
olicy
NAT
lartibartf
mmer Time
General | SysInfo | Compile / Install | Firewall | Network

Version:
Options marked (*) require patch-o-matic

Global Logging Parameters

use LOG
log TCP seq. numbers
log TCP options
log IP options
Use numeric syslog log levels
Log Level: 4: warning
use ULOG
cprange: 0
queue threshold: 1
Netlink Group: 1
Log Prefix RULE %N -- %A
Log all dropped packets (*)
Logging Limit: 0
Turn logging ON on all rules (overrides rule options, use for debugging)

Default Action on 'Reject'
ICMP port unreachable

Options
Turn off code optimisation in compiler
Load modules
Verify interfaces before loading firewall policy
Assume firewall object is part of 'Any'
Clamp MSS to MTU (*)
Accept TCP sessions opened prior to firewall restart
Accept ESTABLISHED and RELATED packets before first rule
Bridging firewall
Detect rule shading in policy
Ignore empty groups in rules

Script Options:
Turn on debug output in iptables script
Create virtual addresses for NAT rules

Figure 5. Platform-Specific Firewall Properties

In the Global Logging Parameters section you can control how Firewall Builder writes log entries. The default Log Level of 6: Info is okay. Personally, I log only dropped and rejected packets, so I like to bump this up to 4: Warning.

In the Options section of the Firewall screen you may wish to select Assume firewall object is part of Any. The default is for the built-in Source/Destination object Any to be interpreted as “Any host except the firewall”. This is not atypical in firewall policy builders, but it can cause some surprising behaviors.

For example, if the last rule in your policy is a cleanup rule that sets source=any, destination=any, service=any, action=drop and logging=on, you'd expect any attempted connections to the firewall not matching previous rules to be logged and dropped, right? Indeed, they will be dropped, but not because of this rule. They'll be dropped by the INPUT chain's default policy, which Firewall Builder always sets to DROP. This example cleanup rule is triggered only by attempted connections *through* the firewall. As the firewall itself is not assumed to be part of Any, your cleanup rule is implemented only in the FORWARD chain, not in the INPUT or OUTPUT chains.

Selecting Assume firewall object as part of Any reverses this behavior, and it causes such a cleanup rule to behave the way you'd expect. However, it may complicate other things, such as anti-spoofing rules specific to firewall interfaces. In short, it's a trade-off. My own preference is to leave this option deselected. Then, I either tweak my Firewall Builder scripts to include the log and drop lines to at least the INPUT chain, or I add an extra Firewall input log and drop rule to my policy.

If in doubt, test and tinker with this setting. You can use the Log all dropped packets in the Global Logging Parameters section, but it requires your firewall to have had Netfilter compiled with the Patch-O-Matic Dropped Table patch. This may not be the case if you installed a kernel provided with your Linux distribution.

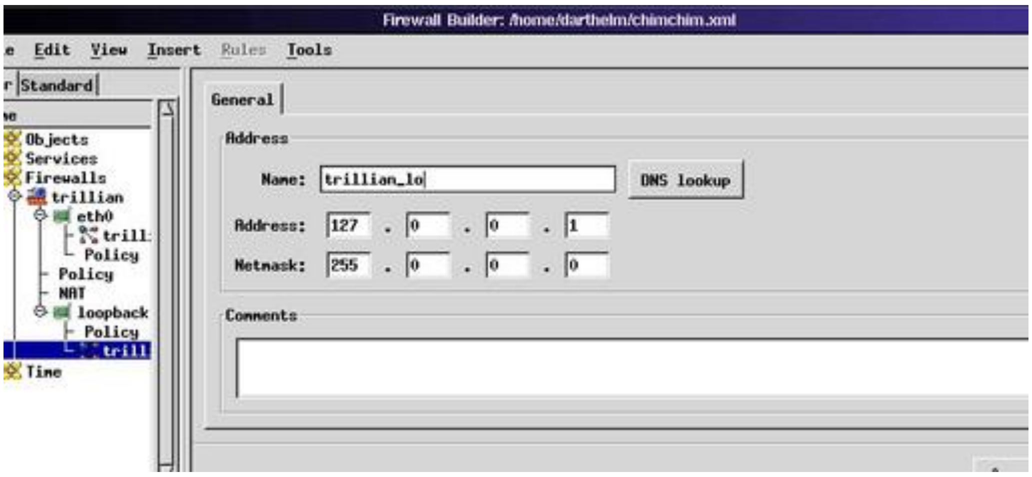
The last screen of Firewall object properties is Network. This contains settings specific to the Host OS you specified in the General screen. These options directly alter your kernel's behavior; if that frightens you, ignore this screen. But if your firewall is a full-blown, entire-network-protecting firewall rather than simply a hardened host, make sure you set Packet Forwarding to On.

Loopback Interfaces

Believe it or not, even after all this work we're not done configuring the firewall object. In Figure 4 you may have noticed that in the hierarchical view of objects on the left-hand side of the window, the firewall Trillian has two interfaces, eth0 and lo. The eth0 subsection was created automatically when I ran the Insert firewall dialog. lo, which represents Trillian's loopback interface, had to be created manually. It's a little odd that its creation doesn't happen automatically. Every firewall, whether a multihomed host or a bastion host, needs rules that allow its loopback interfaces free rein (so local processes aren't interrupted).

To add a loopback interface manually, select your firewall object's icon in the object list, pull down the Insert menu above and select Interface. The Interface option is grayed-out unless you've selected a Host or Firewall object. A new interface icon appears below your firewall object, and the new interface's properties are displayed on the right. Enter the interface's name into Name (for example, lo), and deselect This Interface is External (insecure). The latter option should be checked only for your external interface and DMZ interface objects.

Next, while the new interface's object is still selected, pull down the Insert menu again and select Address. Now an address sub-object appears below your new interface, and its properties are displayed on the right (Figure 6). Enter a name, an IP address of 127.0.0.1 and a netmask of 255.0.0.0 (the latter will be filled in automatically, actually). In some circumstances, systems have more than one loopback interface, in which case the address may vary (127.0.0.2, etc.). Chances are, though, you have only one and its IP is 127.0.0.1. When in doubt, do an **ifconfig -a** on your firewall.



Name: trillian_lo
Address: 127 . 0 . 0 . 1
Netmask: 255 . 0 . 0 . 0

Figure 6. Adding a Loopback Interface's Address

Once you've defined all your objects, or at least enough to start writing rules, save them by pulling down the File menu and selecting Save. You'll be prompted to provide a filename with a suffix of .xml in your home directory. Some scripts expect objects to be named objects.xml and to be stored in ~, but this is user-configurable. In other words, name your objects file whatever you like and put it wherever you want. Remember what and where these are if you need to tweak fwb_install or another policy installation script.

Next Steps, Next Month

I'll leave it to you to create additional Host, Network and Firewall objects pertinent to your environment. Also, I've skipped Network Range and Time objects, both of which are easy to understand and use—you can figure these out by playing with them or by referring to the documentation at www.fwbuilder.org. Next month, we'll use all these objects to build some policies. In the meantime, I hope you've learned enough to get started and to start exploring Firewall Builder on your own. Have fun!

Resources

Mick Bauer (mick@visi.com) is a network security consultant for Upstream Solutions, Inc., based in Minneapolis, Minnesota. He is the author of the O'Reilly book Building Secure Servers with Linux, composer of the “Network Engineering Polka” and a proud parent (of children).

Archive Index Issue Table of Contents

Advanced search

Closing the Chasm

Doc Searls

Issue #109, May 2003

Linux already has bridged the product gap between early adoption and the mainstream. Now the big challenge is cultural.

In most big bookstores, the books you see first don't last. They're out on the front tables for you to sample. If they sell well, there's no guarantee they'll stay on the shelves for months, much less years. Even the biggest bookstores don't have enough room to store a fraction of the new books that wash in and out, like foam on a tide.

Those that stay are part of the culture. You expect to find them at a bookstore because their appeal endures. With fiction, the appeal may last years, decades or centuries. Nonfiction books, however, bear the burden of relevance. Among the nonfiction categories, business books don't age as rapidly as sports and travel titles, unless they're about technology. Books about technology trends and companies tend to age as well as last week's meat.

When I look at my bookshelves, I wince at titles like *The Big Tech Score* and *Managing Inter@ctivity*. It's not that any of these books contain bad information; they simply speak of a time that is going or gone.

One exception is the work of Geoffrey A. Moore. Although his books are packed with examples from companies whose names are no longer in use, his insights about technology adoption remain relevant, especially (and ironically) for Linux. In fact, I think Moore's technology adoption model is a handy way to make sense of Linux's quietly growing success inside large enterprises. It also will be useful when it is time to grok Linux's inevitable success on desktops as well.

Moore's model is an old one: the adoption curve, which dates back to Everett M. Rogers' *Diffusion of Innovations*, first published in 1962 and still going strong, broken into pieces (Figure 1).

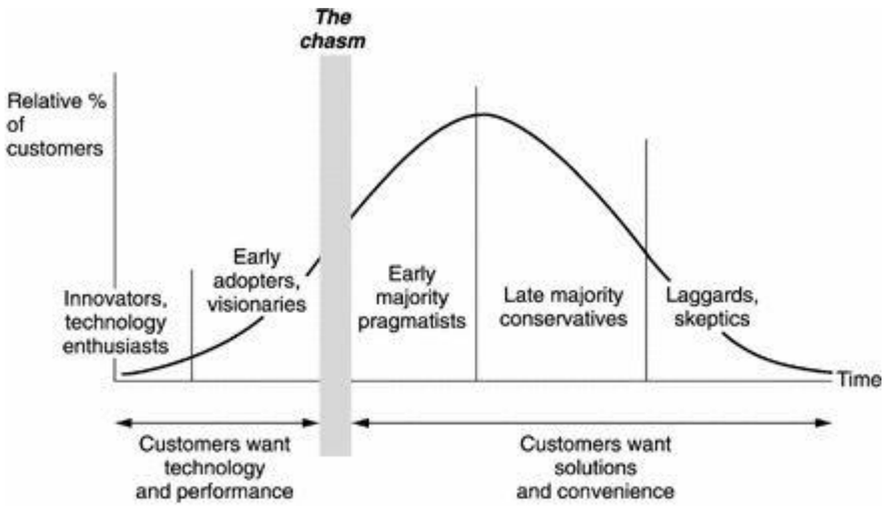

The chasm
Relative % of customers
Innovators, technology enthusiasts
Early adopters, visionaries
Early majority pragmatists
Late majority conservatives
Laggards, skeptics
Time
Customers want technology and performance
Customers want solutions and convenience

Figure 1. The Adoption Curve

His main focus is the chasm between early adopters and the early majority. In *Crossing the Chasm* (1991) and *Inside the Tornado* (1995), Moore described the width of the chasm in terms of cultural opposites. According to Moore, the techie innovators and visionary early adopters on the left side of the chasm are radically different from the pragmatic early majority (Table 1).

Table 1. Cultural Opposites [from *Inside the Tornado*, page 18]

What appeals to both groups is also radically different. Techies and visionaries care about product qualities such as speed, design, ease of use, price, novel features and functions, impressive demos and trade-press coverage. Pragmatists care about reliable solutions, third-party support, de facto standards, cost of ownership, quality of support and success of colleagues.

Notice something funny? Did Linux skip a generation here? As a product, isn't Linux almost profoundly pragmatic? And, isn't it strange that what made Linux's visionaries excited was a roster of mostly pragmatic values?

The graph in Figure 1 isn't Moore's; it's Don Norman's version of Moore's curve. Dr Norman (www.jnd.org), a scientist whose dozens of books include such classics as *The Design of Everyday Things*, *The Invisible Computer* and *Things That Make Us Smart*, is a fan of Moore and his books. About *Crossing the Chasm* he says (www.jnd.org/dn.mss/life_cycle_of_techno.html):

> The classic marketing book for high-technology companies, widely read and discussed, but almost never followed. Why is it so difficult for a high-technology company to understand that late adopters of a technology are very, very different from the technology enthusiasts who made the company successful? Because the whole culture of the company is based on its wildly successful teenage years, and high-tech companies hate to grow up. Immaturity is

> embedded in the culture. Technology is easy to change. Culture is hard.

But the culture Moore and Norman talk about is on the supply side—it's the culture of companies who make their living selling technology. It's different with Linux. “Linux company” has always had an oxymoronic quality. Lately I've begun to think there never has been such a thing as a “Linux company”. Linux is too deep, too infrastructural, too free. Yes, you can productize and brand it, just as Pepsi productizes filtered water and sells it as Aquafina. But, like air and water, Linux is too elemental to be a product in itself. Here's how Don Norman puts the distinction:

> There is a big difference between infrastructure products, which I call non-substitutable goods, and traditional products, substitutable goods. With traditional goods, a company can survive with a stable, but non-dominant market share. Coke and Pepsi both survive. Cereals and soaps have multiple brands. With infrastructure goods, there can be just one. MS-DOS won over the Macintosh OS, and that was that. MS-DOS transitioned to Windows, and the dominance continued. VHS tape triumphed over Beta. Most infrastructures are dictated by the government, which assures agreement to a single standard. When there is no standard, as in AM stereo or digital cellular options in the US, there is chaos.

He's right—if we're talking about the present moment in time. But reports are coming in from rank-and-file infrastructure builders at the world's biggest technology companies. At FedEx, Boeing, General Electric, Nokia, Sony, Matsushita, Philips and the rest of them, Linux is becoming the infrastructural standard. It's not killing off Windows; it's simply supplanting it as infrastructure. Where Windows persists, it's being repositioned as a “platform” rather than as a standard. In Don Norman's terms, Linux is becoming non-substitutable while Windows is becoming substitutable.

When Don Norman wrote that paragraph, DOS/Windows was de facto infrastructure. What's happened since then is what many in our community have expected for a very long time: free forms of UNIX eventually will find adoption as universal infrastructure. Sure enough, today most web servers run on Linux or BSD. Sony, Matsushita, Philips and a pile of other consumer electronics giants are codeveloping a free and open embedded Linux distro for their own future product generations. DOS/Windows, which quietly used to serve as the OS for countless cash registers and point-of-sale (POS) terminals, is being replaced rapidly by new models that run on Linux.

In *Inside the Tornado*, Geoffrey Moore subdivides early adoption into two stages: The Bowling Alley and The Tornado. The Bowling Alley is “a period of niche-based adoption in advance of the general marketplace, driven by compelling customer needs and the willingness of vendors to craft niche-specific whole products." The Tornado is "a period of mass-market adoption, when the general marketplace switches over to the new infrastructure paradigm." These are followed by Main Street, "a period of after-market development when the base infrastructure has been deployed and the goal now is to flesh out its potential."

For web servers, database servers, rendering farms, point-of-sale systems and a variety of other niche categories, Linux has been bowling strikes for several years now. Now we're moving inside the tornado, big time.

But it's a stealth tornado, because few companies on the supply side (IBM is a huge exception) even bother to make a big thing about it. The real leadership is happening on the demand side, among the pragmatists.

Here's Linux. It's lying around looking useful. A lot of technologists know their way around or can learn it easily. It's free. There's no vendor lock-in. There are no royalties or other fees to pay. It's extremely useful as building material. There are lots of tools you can use to build with it, and most of those are free too. The only problem is PR; it still looks like it's a Visionary Thing.

But that image fits a pattern too. This one is described by Clayton Cristensen in another classic, *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail* (Harvard Business School Press, 1997). Cristensen says Great Firms are innovative by nature (witness Microsoft's defensive talk about their right to innovate), but their innovations are incremental. They improve the proven in gradual steps; they don't tip over their cash cows—and for very good reasons.

One is dependence on customers and investors as sole resources. Cristensen says:

> While managers think they control the flow of resources in their firms, in the end it is customers and investors who dictate how money will be spent because companies with investment patterns that don't satisfy their customers don't survive. The highest-performing companies in fact are those that are best at killing ideas that their customers don't want. As a result, these companies find it very difficult to invest adequate resources in disruptive technologies—lower-margin opportunities that their customers don't want—until their customers want them. And by then it's too late.

Other reasons for avoiding disruptive technologies are “small markets don't solve the growth needs of large companies” and “markets that don't exist can't be analysed”. On that last point, Cristensen says:

> Because the vast majority of innovations are sustaining in character, most executives have learned to manage innovation in a sustaining context, where analysis and planning were feasible. In dealing with disruptive technologies leading to new markets, however, market researchers and business planners have consistently dismal records. In fact, based upon evidence from the disk drive, motorcycle, and microprocessor industries, the only thing we may know for sure when we read experts' forecasts about how large emerging markets will become is that they are wrong.

All disruptive technologies, Cristensen says, are not taken seriously at first. They look like toys. They seem underpowered. The wrong people are selling and adopting them. They seem to be missing crucial features and functions. They don't fit anywhere in current product lines. They're not “standard”. And because they start out that way, disruptive technologies remain ignored, underestimated and misunderstood, even while they continuously improve. That's why customers discover disruptive technology advantages ahead of the “innovative” vendors whose gradually improving goods are suddenly made obsolete.

Linux is even more disruptive than any of Cristensen's examples, because its primary movement isn't from companies to customers—it's from hackers to customers. The creators of free tools, components and applications make the products of their labors available to anyone. In this respect, Linux resembles such natural building materials as rocks, wood, iron and concrete. The difference is that Linux occurs in human nature.

So companies don't “adopt” Linux so much as they mine and harvest it. Only, because it's digital stuff, there's no scarcity. That makes it infinitely less expensive than cheap lumber, rock, minerals and petroleum. All of which make Linux that much easier to adopt and that much more disruptive.

The remaining challenge, then, is cultural. There the chasm remains intact. On one side we have a bunch of techies who care deeply about the principles that brought Linux into the world and that made it so useful to so many at so little cost. They get worked up over ethical issues and about licenses like the GPL, which respects the nature of Linux and its development while hardly caring at all about its commercial potential. On the other side we have a bunch of techies who care deeply about solving problems and making things work, who hardly care at all about the political and moral issues that get the first side so worked up.

In fact, the two sides aren't opposed. They're just different constituencies with different priorities. The market logic is *and*, not *or*. We'll know we've bridged that chasm when we stop making a big deal about it. At that point we'll all be on Main Street.

Doc Searls is senior editor of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Doing Good and Preventing Bad

Phil Hughes

Issue #109, May 2003

You can help by fighting legal FUD or by doing useful work on Linux.

Thirty years ago I was working at the Hanford Nuclear Reservation. The position focused on systems programming, but ultimately, the environment rubbed off on me. I learned a lot about nuclear power as well as the more antisocial aspects of nukes. And, I recognized a problem. Any number of issues came to mind: operational danger, waste disposal and life-cycle cost are the first three. In any case, I knew nuclear power was not going to be, as President Eisenhower had said 20 years before, “too cheap to meter”.

I wanted to make sure everyone learned what I had regarding the issues of nuclear power. But, I also wanted everyone to learn all that I had learned about alternative energy sources such as solar, biomass and wind.

I quickly found I could not deal with both issues. Understanding and teaching about alternative energy is a huge job, as is pointing out flaws in nuclear power. So, I chose. Because of all I had learned at Hanford, I felt I was more qualified to talk about the problems of nuclear-power generation.

This was my choice for “preventing bad”. The more knowledge I could disseminate, the more likely it would be that the general population would see the issues, get involved and, in the long term, prevent the US from jumping deeper into the nuclear well.

Unfortunately, new “bad” appeared. As photovoltaics became cheaper, utility companies lobbied to make it harder for customers to sell power back to the grid. Once it was proved that intertie systems to sell power back to the utilities were safe and effective, utilities came up with rate schedules where the power you sold back was at a lower price than the power they sold you, even though peak output of solar systems corresponded with very high demand. They wanted to use the law to escape the consequences of failure by making intertie systems pay to decommission failed nuclear plants.

Now, back to software. When I first saw Linux (back when kernel versions started with a dot), I merely was looking at alternatives to the “real” UNIX systems. We had been in business publishing pocket reference cards and doing training and consulting on UNIX systems for about ten years.

I decided Linux was a lot more than simply a hobby project. I felt it showed great promise, so we changed direction from being a UNIX company to being a Linux company. With over 100 issues of *Linux Journal* under our belt, I feel we made the right choice.

Early on, a lot of my energy went into telling people about the virtues of Linux. Much like telling people that nuclear power costs too much, this was a hard sell at first. People didn't want to hear that Linux might be a better choice than what they were working with.

Now the days of telling people that Linux is a serious contender in the OS business are gone. Even if you still don't have it on your desktop, it is unlikely you will sit down for a web surfing session without getting involved with some Linux server.

The problem is that much like solar panels on your roof are a threat to the nuclear power industry, Linux is a threat to the OS status quo. One can spend a lot of time and energy counteracting the FUD.

As Linux is proving to be a worthy alternative, the same sort of “it works but you can't put it here” arguments are appearing as they did with intertie systems. They merely go by different names: DMCA and bogus software patents are two of those names. As with nukes and solar energy, competition is fine, but what we see is the use of the legal system to harass promising new alternatives.

This work of “preventing bad” needs to be done. But, the alternative is simply to keep on truckin' with Linux—continuing to identify places where Linux solves a problem and moving ahead with the solution. So many places have either a time-consuming manual system or a poorly implemented, non-Linux system that people easily can make a career of problem solving with Linux.

The Linux movement needs both. Someone has to deal with FUD, and someone needs to move Linux into new places. To go back to my nuclear power analogy, if no one were out there developing alternative energy technology, there would be no alternative to nukes, no matter how bad a picture the antinuke activist painted.

As for me, I did my time dealing with the FUD. A lot of it was fun work, but I have pretty much moved into the “just do it with Linux” camp. I would rather show someone a solution and let them choose than spend my time counteracting anti-Linux propaganda.

I moved to Costa Rica over a year ago. On my one-year anniversary I was thinking about how things have been different. Aside from the more obvious—things like I really need to be working on my Spanish—the biggest difference I have seen is that people are more open to solutions. There is less disposable money here than in the US and less anti-Linux FUD. Thus, it is easier here to listen to someone's problem, propose a Linux-based solution and have them accept it than in the US. It is easier to think Linus was right about World Domination—except that the US might be the last country to get Linux.

By the way, besides being Linux-friendly, all electricity in Costa Rica is produced from renewable sources including hydro, geothermal, wind and solar. Maybe these two issues fit together more than I thought.

Phil Hughes is the publisher of *Linux Journal*.

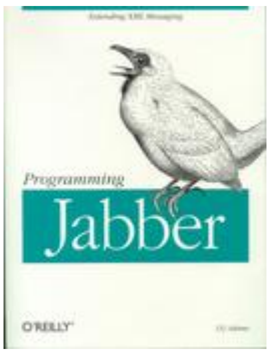
Programming Jabber

Paul Barry

Issue #109, May 2003

Adams' greatest feat is proving Jabber to be much more than an IM system by concentrating on the infrastructure provided by the technology.

Book Review: *Programming Jabber* by D. J. Adams



O'Reilly & Associates, Inc., 2002

ISBN: 0-596-00202-5

$39.95 US (hardcover)

I've always been intrigued by the idea of using Jabber as a router for XML messages, which happens to be the topic of D. J. Adam's *Programming Jabber*.

Things start out well enough. The first example program appears on page 7, and even better, it's in Perl. Unfortunately, the next example program, a script for registering new Jabber users, appears 221 pages later. In between is a detailed description of the Jabber protocol and gobs and gobs of XML. This material is authoritative, but I wonder whether an appendix or two wouldn't have been a better place for much of it. This critique is especially true for the descriptions of the Jabber.xml configuration file in Chapter 4 and Jabber namespaces in Chapter 6. In addition, this material is tough going. The book is written in the style of a reference guide, and it's pretty dry.

Thankfully, the remaining chapters (7-10) are the book's salvation. This material describes programming Jabber and its protocol from a number of perspectives. Not only does Adams provide explanations for writing a series of Jabber clients, but he also shows readers how to extend Jabber with a custom component. There's even a program that interfaces with LEGO MindStorms to determine whether there's coffee in the pot.

The book covers three programming languages, Java, Python and Perl, with Perl receiving the most coverage. With each of the languages, I would have welcomed brief coverage of the libraries Adams relies on in his code. Granted, it's a book about Jabber's protocol, but it would be more complete if it included further information on the various Jabber libraries.

Adams' greatest feat is proving Jabber to be much more than an IM system by concentrating on the infrastructure provided by the technology. The book covers release 1.4.1 of Jabber. The most recent stable release is 1.4.2 (as of December 2002), so the book is highly relevant to the current Jabber. With release two of Jabber now in alpha, a second edition will be needed soon. For now, *Programming Jabber* is a resource that Jabber programmers won't want to be without.

—Paul Barry

email: paul.barry@itcarlow.ie

Archive Index Issue Table of Contents

Advanced search

Free Software, Free Society: Selected Essays of Richard…

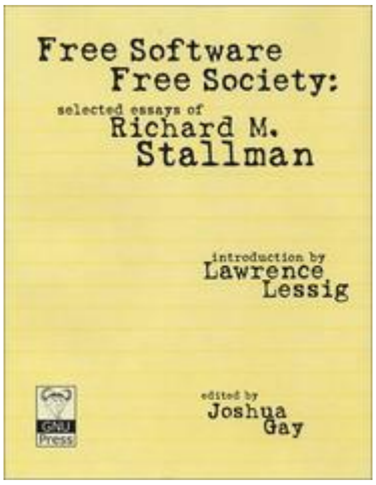
Marco Fioretti

Issue #109, May 2003

Most of this material is available on-line, but there are a couple reasons to buy the book.

Book Review: *Free Software, Free Society: Selected Essays of*

Richard M. Stallman



Free Software Foundation, 2002

ISBN: 1-882-11498-1

$24.95 hardcover

This collection of 21 essays written by Richard M. Stallman between 1984 and 2002 ranges from historical milestones, including the GNU Manifesto, to transcripts of some recent speeches. The introduction is by Lawrence Lessig, Professor at Stanford Law School.

Most of this material is available on-line, but there are a couple of reasons to buy the book: First, the profits go to the Free Software Foundation. Second, having all the essays in one place, with cross-references, updates and notes from Stallman himself, helps readers see the big picture.

Lessig's introduction provides the right start by pointing out that if the Free Software movement is new, radical or revolutionary, it is because it brings to software the freedoms already present in the pre-software world. For example, laws and legal briefs are Free as in Freedom.

The grouping of the essays shows how Stallman himself and the movement in general have been forced to evolve over time. In the beginning it was “only” about the freedom to program, which a minority of people needed when computers were not widely available.

Today, almost everybody's entertainment, work, education and free speech rights depend on computers. The essays in the second and third parts of the book cover why the DMCA and similar efforts are harmful to citizens' rights and a market economy.

In less than 200 pages, we go from “GNU will remove operating system software from the realm of competition” to the problems of copy-restricted media.

A lot of details on how this evolution happened are provided, with several repetitions. Sometimes, these are even funny: the history of the Xerox printer, whose proprietary driver made Stallman mad enough to start the whole thing, is told so often that one can picture proprietary software executives cursing Xerox for not just giving him the darn code.

Coverage of one fundamental issue, free file formats, is missing. Stallman wrote a 2002 essay called “We Can Put an End to Word Attachments” that addresses this need, but it's not in the book.

In general, the book is necessary reading and not only for programmers. I personally disagree with Stallman on certain conclusions and am still trying to decide whether I accept some others or not. But it is crucial that everyone thinks about these problems today, builds his or her own conclusions and follows them. Even if you reject all Stallman's ideas, you must know why, and this book will help.

—Marco Fioretti

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #109, May 2003

Readers byte back.

Replacing Microsoft Exchange

First off, I really did enjoy reading the article on replacing Microsoft Exchange with Linux [“Understanding and Replacing Microsoft Exchange” by Tom Adelstein, *LJ*, February 2003]. The article didn't explicitly state you were trying to replace Microsoft Exchange 5.5, but it was obvious to me that was the version you were working with. Microsoft's Exchange 5.5 was limited in the total size of the database, but that has changed in Exchange 2000 and Titanium. Exchange 2000's Exchange Storage Engine has been rewritten to allow much larger database sizes, which would be limited only to the total capacity of the hardware on which the database would reside. Also, I fail to understand the reasoning of why IMAP was provided as a connection mechanism. IMAP will provide you with a way to preview the e-mail message. Does your DLL plugin to Outlook provide a way for the message to remain on the e-mail server?

—Chris Lynch, Network Engineer

Tom Adelstein's reply: Thanks for your thoughtful letter. I appreciate your detailed understanding of Exchange. It is quite impressive. Our goal at Bynari was to create an RFC-compliant platform that allowed Microsoft Outlook to run in corporate workgroup mode. In the case of Outlook 2002, we wanted to provide calendar sharing and delegation.

A Bas le Franglais!

I have been a happy *LJ* subscriber for many years now. I am usually not one to complain about things but the articles written by Marcel Gagné are getting irritating. His French Chef schtick is getting VERY old and detracts from his articles. Please consider having him stop that and simply write informative articles.

—David Vohwinkel

Can I Cluster This App?

It is with much interest that I have read your editorial on clustering the previously unclustered [From the Editor by Don Marti, *LJ*, February 2003]. I seem to be unable to convince the powers that be at some CAD and CAD/CAM firms that this is an option. Therefore, I would very much like to get information on how to make programs that are not enabled for cluster technologies to work with cluster hardware configurations.

—Darald

If you have one huge, slow process, you'll probably need to rewrite the software using a clustering library. However, if you need to make many processes cooperate, OpenMosix automatically will migrate some of them to idle nodes in the cluster. See openmosix.sf.net.

—Editor

More on Screen, Please

Thank you for excellent article on screen(1) ["Power Sessions with Screen" by Adam Lazur, *LJ*, January 2003]. Please persuade Adam Lazur to write a couple follow-up articles to this introductory one. I'm already starving for more.

—Jari

Geeklog Isn't a PHP-Nuke Fork

In the March 2003 issue, there was an article on weblogs that led readers to believe that Geeklog is a fork of PHP-Nuke ["Building with Blogs" by Doc Searls and Dave Sifry]. Much respect needs to be given to PHP-Nuke for helping to spur the weblog phenomenon, and I think the article presented that respect. However, Geeklog is not a fork of PHP-Nuke. Also, the article should have mentioned www.opensourcecms.com. It has the most popular packages in a demo-able state for users to test them without having to download, install and configure.

—Tony Bibbs

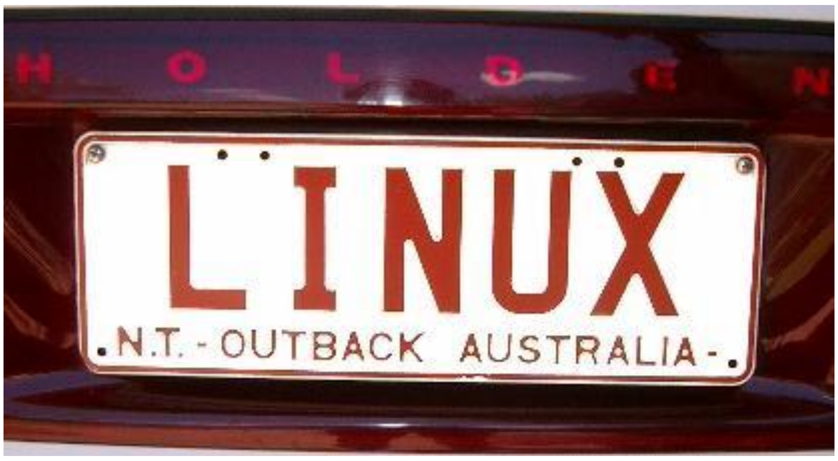
MPAA vs. Free Speech

Seth Schoen's column "Broadcast Flag: MPAA's Latest Attack on Linux" (*LJ*, March 2003) appears to be mistitled. Mr. Schoen did not provide evidence on the MPAA making any form of attack on Linux or on any free operating system in the article. On a side note, he wrote, “Reporters were barred from meetings, which had a $100-per-meeting admission fee.” Were reporters specifically barred or was it just that reporters would have to pay $100 per meeting?

—Jason

Seth Schoen replies: Magazine editors have little space available for titles, so I could see where they would feel the need to condense “Broadcast Flag: MPAA's Latest Attack on the Right to Use Free Software to Lawfully Make and Interoperate with Recordings of Copyrighted Audiovisual Works” (which is what I would have called it) to “Broadcast Flag: MPAA's Latest Attack on Linux”. Reporters were specifically barred from meetings; they tried to participate, and they were kicked out.

Show Us Your Linux License Plate



I was pleased to see the LINUX license plate in your February 2003 issue [Letters], particularly as I have the same five letters on my car! I too was surprised to find that LINUX had not been chosen in my home state (Northern Territory, Australia). Having a LINUX license plate is certainly a talking point around town. I'd like all the owners of LINUX license plates from around the world to send a picture to *LJ*, along with a few words about their particular part of the world.

—Adrian Casey

Linux Journal Named Our Son

A few years ago, when I first subscribed to *LJ*, someone made a mistake and sent two copies of each issue. One was sent to me, and the other was addressed to Eric W. Sattler, which was strange because there was nobody by that name that I'd ever heard of. I laughed at it, and that was that. In 2001, when our son was born, my wife and I were having trouble coming up with a name, and we remembered Eric from *Linux Journal*. It sounded better than anything we had thought of on our own, so Eric it was. He'll be two years old this June, and I wanted to send this mail to thank you for helping us name our son.

—Tom Sattler

BBS Renaissance

In the March 2003 issue, the From the Editor column talks about community and the fear of spam people have when posting on Usenet or web forums. But there is something old that has become new. Cheap hardware and cable modems have caused a renaissance in the BBS scene, and one of the major factors in this is a GNU operating system named Linux. Why choose to eat spam when you can telnet into a friendly BBS?

—Anonymous

Linux at 14,000 Feet

Although Bolivian users held an installfest at 11,000 feet, astronomers at Mauna Kea regularly install Linux at 14,000 feet. At that height we need extra cooling, usually by placing big fans (ten inches at least) near the monitor, which seems to produce the most heat. The lower efficiency of cooling is a big problem at these altitudes; yet, Linux runs well up there too.

—Peter Teuben

John 14:6

I find a T-shirt on the *Linux Journal* web site to be very offensive. The shirt has a typical Christian fish with the words “Linux Saves” inside. I'm surprised the person(s) involved with designing this T-shirt, its advertising and sale aren't offended.

—Norman Clerc

Tech Tips Please

I remember you used to sprinkle little tech tips, command one-liners and other useful everyday kinds of knowledge throughout your magazine. What happened to those? I found them very helpful. I wrote a script called rpmff (RPM File Finder) to search RPMs looking for a specific file and find it useful.

—Doug Wright

Look for rpmff elsewhere in this issue.

—Editor

LJ Web Site Helps You Shop

I bought two products that recently have been featured on the *Linux Journal* web site: a Hawking PN7127P print server [www.linuxjournal.com/article/6509] and a copy of *Eric Meyer on CSS: Mastering the Language of Web Design* [www.linuxjournal.com/article/6618]. Both are great. I appreciate having the articles, the reviews and even the ads. You folks have found the right balance.

—Mike Tarrant

New Blog Software

I just read the article on weblogging by Doc Searls and Dave Sifry [*LJ*, March 2003]. Nice work. And, just to add another option for Linux, there is the Python Desktop Server (pyds.muensterland.org). It is similar in spirit to Radio Userland but runs on systems where Radio isn't available. In combination with the Python Community Server Software (pycs.net), you can get your own community server up with people participating quite fast.

—Georg

Set and Forget Linux App

I recently built a little web application that quickly became production. So quickly in fact, it was only running on the PC running Linux that I used in development. It became the only Linux server in our otherwise all-Windows data center. That was last October. I never did see a new server, but the application just stayed up and worked. Then last Friday, it suddenly went dark. After being yelled at for how unreliable my “toy” was, I finally got one of our Windows Network “Engineers” to admit, “Well, we saw that box running in the data center, but because no one had touched it in months, we figured no one was using it.” I guess I'll call them to reboot it for me every now and then just so they know it's still in use.

—Michael K. Martin, DVM, MPH

Father and Son Linux

I just want to let you know that I have been getting *LJ* for the past few years. It is a wonderful magazine, and I enjoy reading it every month. My son, who is ten years old, has started to read *LJ*, and he finds it enjoyable to read as well.

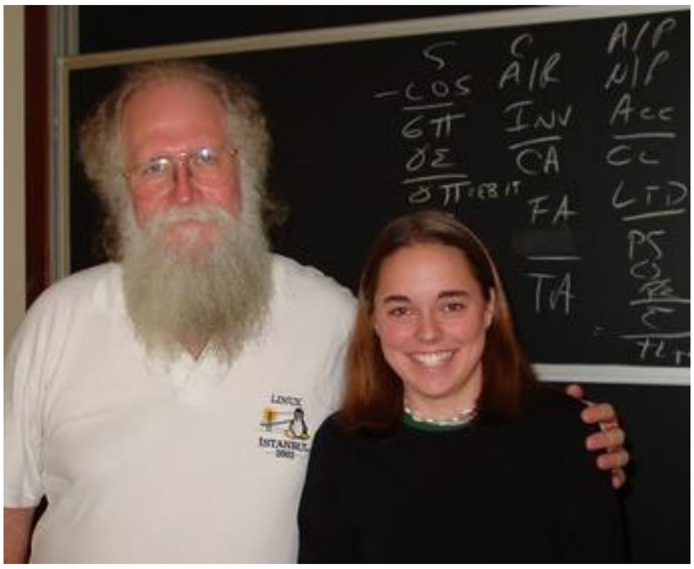
—Jim Wininger

Missing #include

I enjoyed the article about signals [“Linux Signals for the Application Programmer” by Dr B. Thangaraju, *LJ*, March 2003], but let me point a small omission. Listing 4 (page 48) needs to **#include <errno.h>** in order to compile.

—Paul

maddog's Travels



Here's a picture of me and Jon “maddog” Hall at the October 2002 Association of Computing Machinery (ACM) chapter meeting at Northeastern University in Boston. We especially appreciate his regular appearances at our meetings and his colorful and insightful commentary on the future of Linux worldwide. It's through our exposure to people like him that we can form open minds and think for ourselves instead of being led by one company's vision.

—Cathy Swenton, College of Computer Science, Northeastern University, Boston, Massachusetts

Linux Routers Save Hospital Net

I work at a US Navy hospital in Naples, Italy. Recently, my Cisco 7507 inner security screening router died. Then the next week, my core switch. I had to piece together three Linux boxes quickly to use as routers to get through this problem. We are still running on two Linux routers, because my core switch hasn't recovered yet.

—Lee Randolph, CCNP

Sync Files with Unison

rsync [“rsync, Part I” by Mick Bauer, *LJ*, March 2003] is a wonderful tool, but if you want file synchronization, you may not even need to invoke it explicitly. The excellent file synchronization program Unison (www.cis.upenn.edu/~bcpierce/unison) also uses the rsync algorithm.

—Ben Crowell

Freeze, Spammer, You're under Arrest

After reading your comments about spam in the March 2003 issue [From the Editor], I wanted to make a comment of my own. We already have laws that would deal with the jerks who spam. The theft of bandwidth and nuisance could be dealt with through criminal charges, if the government would get off their backsides and enforce the laws we already have.

—Craig Sparks

Give David Bandel More Space

I do not know the details behind the decision to downgrade David A. Bandel's Focus on Software column to the Upfront section of *LJ*, but I see it as a definite loss. Reading Mr. Bandel's column every month was one of the highlights of my subscription.

—Raymond Moczynski

Blog from Emacs

BlogMax is a blog package that's based entirely upon Emacs. It works with both Emacs and XEmacs, is easily extensible and runs on several platforms (billstclair.com/blogmax/index.html). All that is required is a working install of Emacs or XEmacs and access to your web site files via FTP.

—Shane Simmons

Cover Slackware, Please

I'd really like to suggest that *LJ* look into printing information on the Slackware distribution. I've been a Linux user and flag bearer since 1994 when I first installed Slackware; since then I've installed many different distributions of Linux and versions of UNIX. I always come back to Slackware due to its consistency, stability and well-thought-out development. I feel that Slackware has earned a bad reputation as being either dead or underdeveloped, neither of which could be further from the truth! I urge you to please consider an in-depth article on the status and beauty of Slackware.

—David Fleason

Upfront

Various

Issue #109, May 2003

diff -u, *LJ* Index, and more.

diff -u: What's New in Kernel Development

The **devfs** filesystem work by **Richard Gooch** may be coming out of the kernel. At the end of December 2002, **Adam J. Richter** announced a patch to replace devfs with a new mechanism based on **RamFS**. The new system attempted to mimic devfs' behavior in many ways, though Adam did not intend to include all of the devfs functionality in the RamFS system. He wanted his implementation to be, in part, a cleanup of the devfs interface, so that features used only by few systems might be replaced with other methods. As a result of this restructuring, he managed to reduce the size of the code to one-fourth of what it had been. The devfs system always has been controversial, and **Linus Torvalds'** decision to include it in the official tree was even more so. Folks like **Alexander Viro** and others have firmly refused to use it on grounds that it simply wasn't coded well enough. Shortly before Adam announced his own work, Alexander had begun an invasive cleanup and restructuring of the devfs code. Richard, having struggled for years to produce devfs and make it available in timely patches, seems to have vanished entirely from the kernel mailing list.

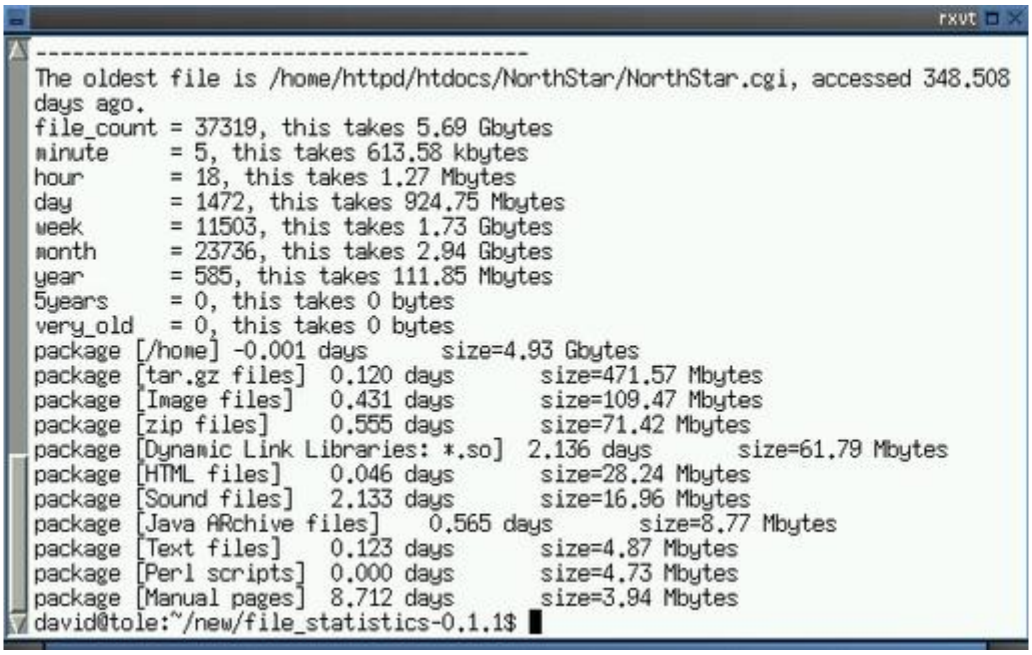
The **sysfs** filesystem is intended to be a replacement for /proc and other methods of exposing kernel data to user space. It began as a tool for driver writers, but its use was broadened in 2002 to all parts of the kernel. Since then, there has been an ongoing effort to migrate a variety of other interfaces to sysfs. In January 2003, /proc/cpufreq came under the knife when **Dominik Brodowski** marked that interface deprecated in favor of a new sysfs interface in the cpufreq core code. **Patrick Mochel** also had a hand in this, making sure Dominik's work matched up with all the latest sysfs features. Later that month, **Stanley Wang** sent some code to **Greg Kroah-Hartman** to replace pcihpfs with a sysfs interface. In this case, however, sysfs was not up to the task as the needed hot-plugging code was not yet fully in place. No problem. Greg coded up the needed sysfs feature and sent it to Patrick.

One day in January 2003, **Alan Cox** happened to mention that the tty code in the 2.5 tree was badly broken and had been for a while, primarily as a result of locking changes in the **kernel preemption** code. This came as a surprise to many people, and some wondered why this was the first they'd heard of it, especially because the 2.5 tree was already in feature-freeze, headed for 2.6 or 3.0. Greg Kroah-Hartman looked at the problem and was horrified. He said it was not going to be easy to fix and was most likely something for the next development tree. But Alan said this wasn't an option, because the tty code was broken already and had to be fixed before the next stable series.

Traditionally, the Linux kernel has been compilable only with the GNU C compiler, and even then it often has been necessary to use a particular version of the compiler to compile particular versions of the kernel. The kernel always has depended on GCC extensions, and the relationship between kernel and compiler has been intertwined for years, like an old married couple. Therefore, various people were shocked to learn that the kernel also could be compiled with Intel's C++ compiler, **icc**. Apparently, Intel has had this as a goal for quite some time, and they've even submitted patches to Linus with the sole purpose of enabling their compiler to handle the kernel source tree.

It's always nice to learn that the feature you desire already has been implemented. According to the documentation (at least as of late January 2003), the only filesystem with **quota** support was ext2. However, apparently work has been going on behind the scenes, because ReiserFS, ext3, UFS and UDF now support quotas.

—Zack Brown



```
----------------------------------------
The oldest file is /home/httpd/htdocs/NorthStar/NorthStar.cgi, accessed 348.508
days ago.
file_count = 37319, this takes 5.69 Gbytes
minute     = 5, this takes 613.58 kbytes
hour       = 18, this takes 1.27 Mbytes
day        = 1472, this takes 924.75 Mbytes
week       = 11503, this takes 1.73 Gbytes
month      = 23736, this takes 2.94 Gbytes
year       = 585, this takes 111.85 Mbytes
5years     = 0, this takes 0 bytes
very_old   = 0, this takes 0 bytes
package [/home] -0.001 days      size=4.93 Gbytes
package [tar.gz files]  0.120 days       size=471.57 Mbytes
package [Image files]   0.431 days       size=109.47 Mbytes
package [zip files]     0.555 days       size=71.42 Mbytes
package [Dynamic Link Libraries: *.so]  2.136 days       size=61.79 Mbytes
package [HTML files]    0.046 days       size=28.24 Mbytes
package [Sound files]   2.133 days       size=16.96 Mbytes
package [Java ARchive files]    0.565 days       size=8.77 Mbytes
package [Text files]    0.123 days       size=4.87 Mbytes
package [Perl scripts]  0.000 days       size=4.73 Mbytes
package [Manual pages]  8.712 days       size=3.94 Mbytes
david@tole:~/new/file_statistics-0.1.1$
```

File access statistics:

www.hszk.bme.hu/~nm127/file_statistics

This utility scans any portion (or all) of the filesystem tree and provides fairly detailed statistics regarding the files on that system. If you happen to be running Debian or a Debian-based system, such as Knoppix, you can receive even more information on the associated dpkg files. This program uses the access times rather than creation or modification times to tell you how "old" or stale a file is. Chances are, files not accessed during the past five years are either historical archives or cruft. Requires: Perl.

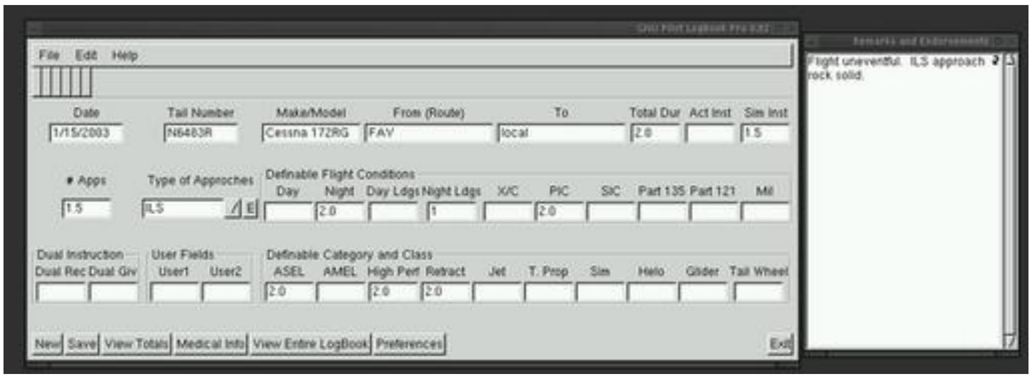
—David A. Bandel

Football Manager: www.autismuk.freeserve.co.uk

Football Manager is a game where you are the manager of a soccer team. Graphics are crude, but the game is a lot of fun. It's a game of strategy where you buy and sell players and choose who will play the game this week. Once you've done your job, sit back for 30 seconds to watch a few shots at the goal and see who won. Then, see your team's rating rise or fall compared with other teams in the league. If I don't remove this game I'll never get any work done—

it's more addictive than *Adventure*. Requires: libSDL, libm, libX11, libXext, libdl, libpthread, glibc.

—David A. Bandel



Date	Tail Number	Make/Model	From (Route)	To	Total Dur	Act Inst	Sim Inst
1/15/2003	N6483R	Cessna 172RG	FAY	local	2.0		1.5

Definable Flight Conditions

# Apps	Type of Approches	Day	Night	Day Ldgs	Night Ldgs	X/C	PIC	SIC	Part 135	Part 121	Mil
1.5	ILS		2.0		1		2.0				

Dual Instruction / User Fields / Definable Category and Class

Dual Rec	Dual Giv	User1	User2	ASEL	AMEL	High Perf	Retract	Jet	T. Prop	Sim	Helo	Glider	Tail Wheel
				2.0		2.0	2.0						

Remarks and Endorsements

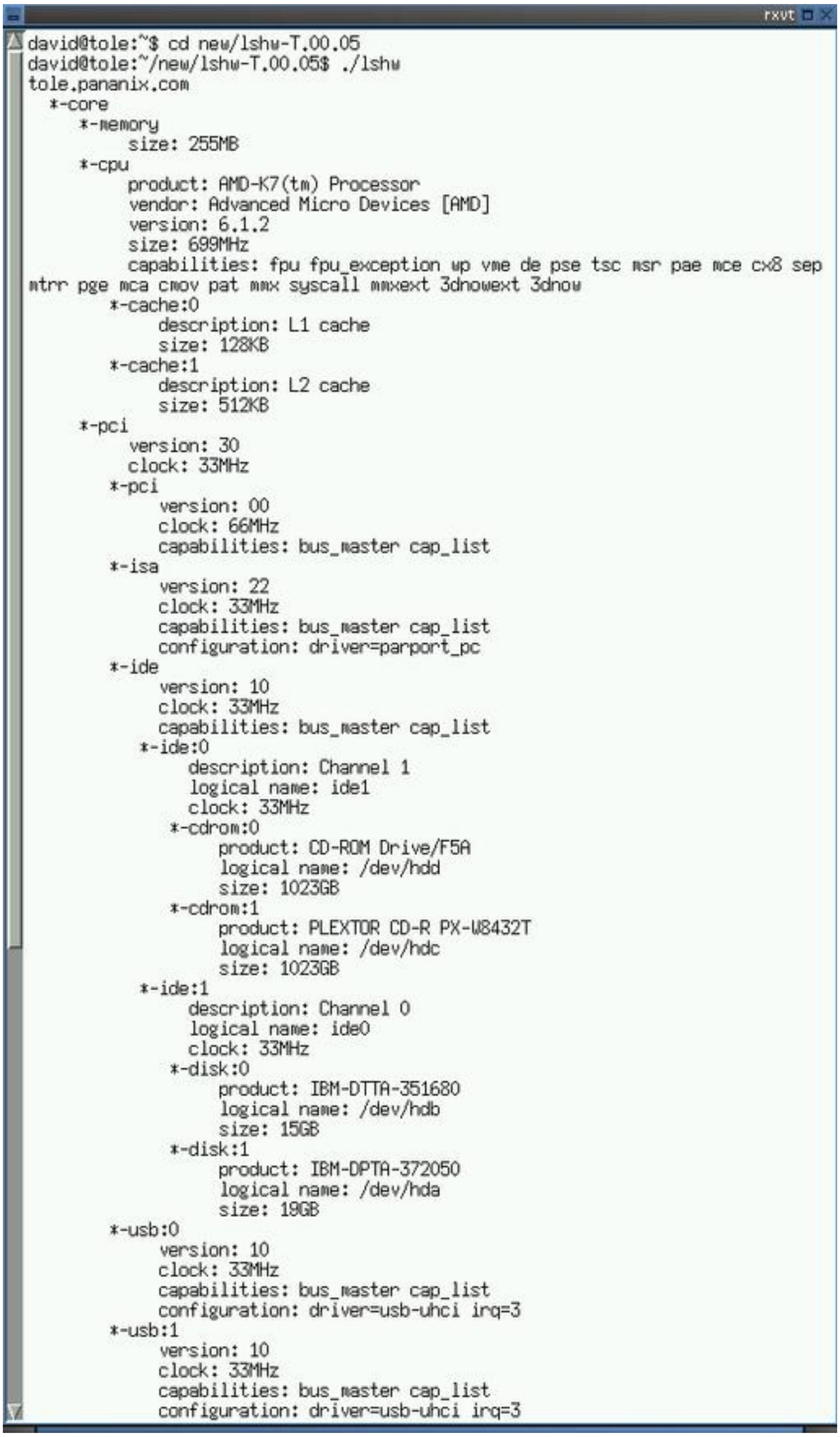
Flight uneventful. ILS approach rock solid.

GNU Pilot Logbook Pro:

ftp.stampede.org/skibum/gplbp/gplbp.tar.gz

If you're a pilot, you know maintaining a logbook is not a big chore. But, when someone wants to know how many hours of which type you have, it becomes a little more difficult. This logbook is like the professional logbook for pilots with all the entries you'll need, plus two user-definable fields. With one click you can see all totals to date. And, by running a small script on the data file (you'll have to create that yourself), you can create a data file for 60 or 90 days back to see how your totals are for currency. Requires: libgtk, libgdk, libgmodule, libglib, libdl, libXi, libXext, libX11, libm, glibc, pilot's license and airplane (last two optional).

—David A. Bandel



```
david@tole:~$ cd new/lshw-T.00.05
david@tole:~/new/lshw-T.00.05$ ./lshw
tole.pananix.com
  *-core
     *-memory
          size: 255MB
     *-cpu
          product: AMD-K7(tm) Processor
          vendor: Advanced Micro Devices [AMD]
          version: 6.1.2
          size: 699MHz
          capabilities: fpu fpu_exception wp vme de pse tsc msr pae mce cx8 sep
mtrr pge mca cmov pat mmx syscall mmxext 3dnowext 3dnow
        *-cache:0
             description: L1 cache
             size: 128KB
        *-cache:1
             description: L2 cache
             size: 512KB
     *-pci
          version: 30
          clock: 33MHz
        *-pci
             version: 00
             clock: 66MHz
             capabilities: bus_master cap_list
        *-isa
             version: 22
             clock: 33MHz
             capabilities: bus_master cap_list
             configuration: driver=parport_pc
        *-ide
             version: 10
             clock: 33MHz
             capabilities: bus_master cap_list
           *-ide:0
                description: Channel 1
                logical name: ide1
                clock: 33MHz
              *-cdrom:0
                   product: CD-ROM Drive/F5A
                   logical name: /dev/hdd
                   size: 1023GB
              *-cdrom:1
                   product: PLEXTOR CD-R PX-W8432T
                   logical name: /dev/hdc
                   size: 1023GB
           *-ide:1
                description: Channel 0
                logical name: ide0
                clock: 33MHz
              *-disk:0
                   product: IBM-DTTA-351680
                   logical name: /dev/hdb
                   size: 15GB
              *-disk:1
                   product: IBM-DPTA-372050
                   logical name: /dev/hda
                   size: 19GB
        *-usb:0
             version: 10
             clock: 33MHz
             capabilities: bus_master cap_list
             configuration: driver=usb-uhci irq=3
        *-usb:1
             version: 10
             clock: 33MHz
             capabilities: bus_master cap_list
             configuration: driver=usb-uhci irq=3
```

Hardware Lister:

ezix.sourceforge.net/software/lshw.html

This hardware lister shows quite a bit of detail, including IRQ, module used and more for cards and other hardware. If you need a great quantity of detail on a system for an inventory, you might want to look at this program. About the only thing missing is the MAC address on the network cards, but that's easy enough to get. Requires: libstdc++, libm, libgcc_s, glibc.

save space.

—David A. Bandel

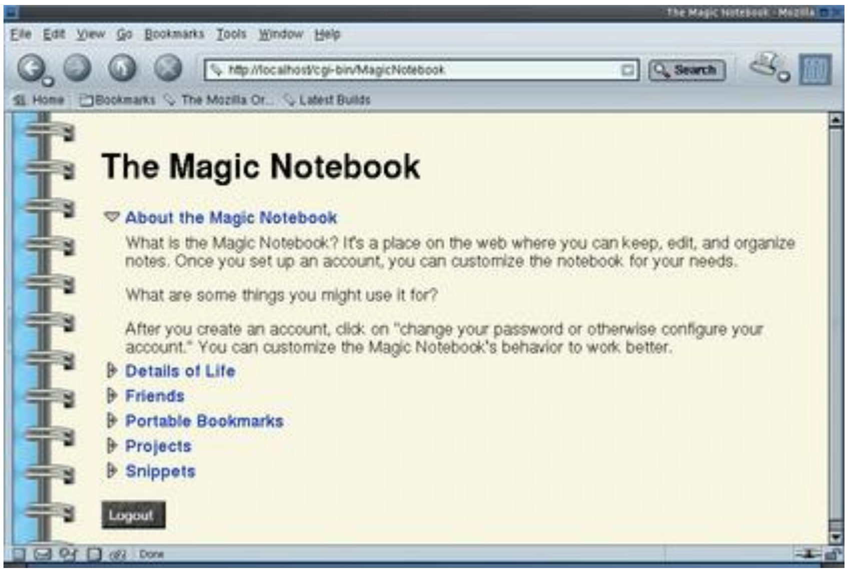
LJ Index—May 2003

1. Bottom price in thousands of dollars of the new SGI Altix 3000 high-end Linux servers: 30
2. Top price in millions of dollars of the new SGI Altix 3000 high-end Linux servers: 1
3. Number of old SGI machines replaced by Dells running Linux at Sony Pictures Imageworks: 600
4. Number of desktop Linux systems now selling at Sams Club's on-line store: 1
5. Price of the desktop Linux systems now selling at Sams Club's on-line store: $297.95
6. Number of different Linux systems (all Microtel) now selling at Wal-Mart's on-line store: 33
7. Number of different Lindows-based systems: 15
8. Number of different Mandrake-based systems: 9
9. Number of different Lycoris-based systems: 9
10. Bottom price for a Linux (Lindows) system at Wal-Mart's on-line store: $199.98
11. Top price for a Linux (Mandrake) system at Wal-Mart's on-line store: $648.00
12. Millions of dollars the Japanese government plans to spend on open-source Linux development for consumer electronics goods in the next fiscal year (starting April 1, 2003): 8.3
13. Thousands of dollars the Japanese government plans to spend in the next fiscal year to study switching its own computers to Linux: 416
14. Position of *Running Linux* among O'Reilly and Associates best-sellers: 1
15. Number of *Running Linux* copies sold: 200,000
16. Current minimum percentage of Linux server shipments, according to Meta Group: 15

17. Current maximum percentage of Linux server shipments, according to Meta Group: 20
18. Linux share of server shipments by 2006 or 2007, according to Meta Group: 45
19. Multiple of performance improvement Reuters Market Data Service gets out of its new Red Hat Linux/HP/Intel systems over earlier proprietary platforms: 2-5

Sources

1-3: *Los Angeles Times*4-5: samsclub.com6-11: walmart.com12-13: Associated Press14-15: "*Running Linux* in a New World" by Russel J. T. Dyer (www.linuxjournal.com/article/6617)16-18: Meta Group, Inc.19: *Wall Street & Technology*


The Magic Notebook - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost/cgi-bin/MagicNotebook
Search
Home Bookmarks The Mozilla Or... Latest Builds
Done

The Magic Notebook

About the Magic Notebook

What is the Magic Notebook? It's a place on the web where you can keep, edit, and organize notes. Once you set up an account, you can customize the notebook for your needs.

What are some things you might use it for?

After you create an account, click on "change your password or otherwise configure your account." You can customize the Magic Notebook's behavior to work better.

Details of Life
Friends
Portable Bookmarks
Projects
Snippets

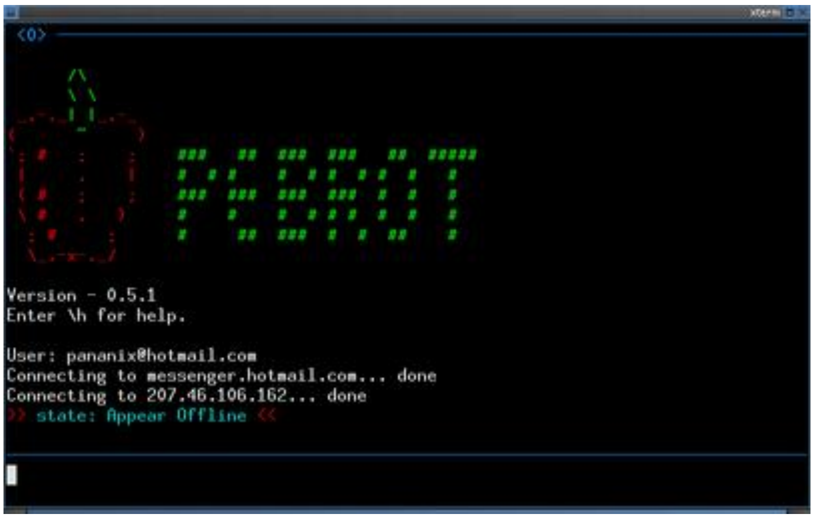
Logout

Magic Notebook:

www.jonathanscorner.com/etc/magic_notebook

Using Magic Notebook is like keeping notes in a notebook, except you use a web interface rather than a pen or pencil. This program can be accessed from anywhere you can reach your web server and can be run normally or encrypted. The notes are stored on your filesystem as HTML files, so if you don't want to use the web interface, the notes are still there. Requires: web server that can serve up CGI scripts, web browser.

—David A. Bandel



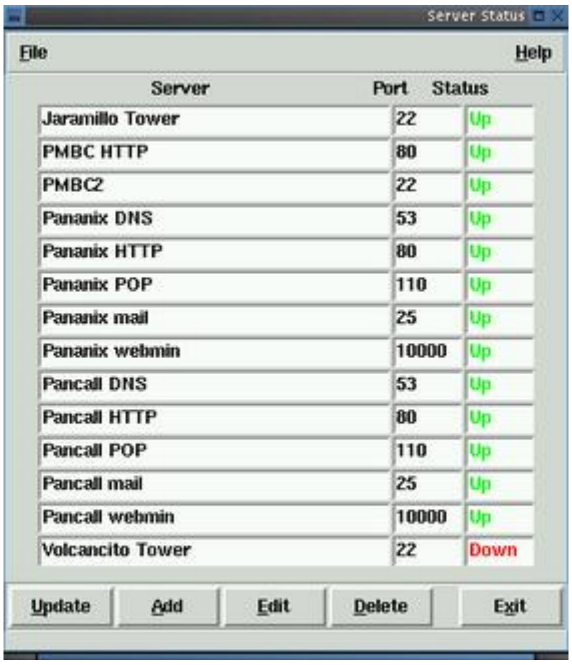
```
Version - 0.5.1
Enter \h for help.

User: pananix@hotmail.com
Connecting to messenger.hotmail.com... done
Connecting to 207.46.106.162... done
>> state: Appear Offline <<
```

Pebrot: pebrot.sourceforge.net

I work on a number of servers and don't install X on most of them, so I'm always looking for command-line programs that can replace X programs. Pebrot is a Python version of MSN Messenger that runs without X, like the UNIX talk program. This makes things easier if you find it necessary to run the program remotely or don't have X installed. Requires: Python.

—David A. Bandel



Server	Port	Status
Jaramillo Tower	22	Up
PMBC HTTP	80	Up
PMBC2	22	Up
Pananix DNS	53	Up
Pananix HTTP	80	Up
Pananix POP	110	Up
Pananix mail	25	Up
Pananix webmin	10000	Up
Pancall DNS	53	Up
Pancall HTTP	80	Up
Pancall POP	110	Up
Pancall mail	25	Up
Pancall webmin	10000	Up
Volcancito Tower	22	Down

Server Status: www.the-den.org/status

This is yet another program that allows you to keep an eye on servers and their status. Although it does require X, it's clean, fast and simple. A number of such programs require SNMP; however, this requires only basic network services. You can leave it running on any system with Perl and Tk and see instantly if you have a problem with a critical service. It won't send you e-mail, but it automatically updates every 60 seconds (configurable) and is easy to read.

Requires: Perl, Perl modules IO::Socket, Tk, Tk::Checkbutton, Tk::Menubutton, Tk::Optionmenu.

—David A. Bandel

They Said It

The problem with intellectual property law is that it tries to take something that is extremely difficult to define and put hard definitions around it. It's not a system that we want to try to embed in cyberspace in the early days of this development....We're creating the architecture, the foundation for the social space where everybody in humanity is going to gather. And if we jigger the foundation design to suit the purposes of organizations that will likely be dead in 15 years, how shortsighted is that?

—John Perry Barlow

It's hard to find successful adults now who don't claim to have been nerds in high school.

—Paul Graham

Linux servers are taking on new roles in enterprise computing, moving from the web-centric workloads, where they are already well established, and moving into application-serving and database-serving workloads. This move is being made possible as ISVs (independent software vendors) port more applications that formerly had been running only on UNIX servers and Windows servers to Linux servers. IDC expects that Linux servers will continue to evolve, both in “scale out” clustered configurations for technical and commercial computing and in “scale up” configurations for larger databases with a single-system image.

—Jean S. Bozman, research vice president of IDC's Global Enterprise Server Solutions Group

Ericsson Releasing TIPC to Open Source under GPL: One Step Ahead toward Building Carrier-Grade Linux Clusters

Ericsson released the source code for TIPC (Telecom Inter-Process Communication) to the Open Source community on February 3, 2003 under the GNU General Public License (GPL). TIPC is a specially designed protocol for intracluster communication and has been used as a part of Ericsson products for years, deployed at hundreds of sites around the globe. It is now ported to Linux and is implemented as a loadable kernel module.

TIPC is a useful toolbox for anyone wanting to develop or use carrier-grade Linux clusters. It provides the necessary infrastructure for cluster, network and software management functionality.

The functional addressing scheme seems to be unique, as is the subscription services and agile connection concept. The signaling link implementation, providing full load sharing and safe failover over any type of bearer is also an asset.

TIPC features include:

- Full location transparency: TIPC provides a functional addressing scheme, hiding all aspects of the cluster's physical topology for the application programs. Mapping between functional and physical addresses is performed transparently and on the fly using a distributed, internal translation table.
- Lightweight, “agile” connections: by avoiding any hidden protocol messages, the message exchange within a transaction, including connection setup, short data transfer and shutdown, can be tailor-made by the user and, hence, be made more efficient. An established connection will react to and report a problem to the application upon any kind of service failure.
- Generic, adaptive, signaling link protocol: tasks that typically are implemented in the transport layer, such as retransmission, segmentation, bundling and continuity check, are pushed down to the signaling link layer. This makes the link layer more complex but provides better resource utilization and results in a more efficient stack. Signaling links are tightly supervised by a continuity check of configurable frequency and are able to detect and report link failures within a fraction of a second. Failover to redundant links in such cases is handled transparently and is disturbance-free. Signaling links are self-configuring, using a broadcast/multicast neighbour-detection protocol when possible.
- Performance: TIPC transfers short (< 1KB) single messages between processors, 25-35% faster than TCP/IP and with comparable speed for larger messages. For intraprocessor messages, delivery speed is 75% better. Furthermore, by using the lightweight connection mechanism, a transaction can be performed by exchanging as few as two messages, to be compared with a minimum of nine in TCP/IP. Hence, short transactions, typical in telecom applications, can be performed in a fraction of the time of corresponding TCP transactions.
- Quality of service: in-sequence, loss-free message delivery can be guaranteed in both connection-oriented and connectionless mode. In case of destination unavailability, nondelivered messages are returned to the sender along with an error code indicating the cause of the problem.

- Subscription services: it is possible for application programs to subscribe for the availability/non-availability of functional and physical addresses. This means it is easy to keep track of both functional and topological changes in the cluster, as well as to synchronize the startup of distributed applications.

We are planning to write a full technical article on TIPC for *LJ* in the upcoming months; meanwhile, feel free to contact Jon Maloy (Jon.Maloy@Ericsson.com) to discuss any aspect of TIPC.

Resources

- Carrier Grade Working Group: www.osdl.org/projects/cgl
- Open Source Development Lab: www.osdl.org
- TIPC Web Site at SourceForge: tipc.sourceforge.net

—Jon Maloy and Ibrahim Haddad

email: david@pananix.com

Archive Index Issue Table of Contents

Advanced search

Linux 2.6: Can You Break It?

Don Marti

Issue #109, May 2003

A preview of the 2.6 kernel and the audio excellence of Linux.

As this issue goes to the printer, kernel development is in feature-freeze. Pretty soon comes the fun part—testing. Almost as important as the work the kernel developers do is the work the kernel testers do. And you don't have to be a kernel expert to test the pre-2.6 kernels on your hardware or your application. By the time you read this, your favorite Linux news sites and mailing lists already might be getting “Please test” messages. Watch your favorite Linux sites, including www.linuxjournal.com, for news about the kernel testing process and how you can help.

Can Linux do for music what it did for server applications? Ask Dave Phillips. His linux-sound.org web site is a resource for everything from hard-disk recorders to music notation editors. This month, he's written a massive roundup of softsynths, software that uses a bewildering array of techniques to turn your Linux box into a synthesizer studio. Plug in a MIDI keyboard or run a sequencer, and follow along on page 80.

If you practice, soon you'll be listening to music you made, and it will sound better than CDs you buy from the major labels. Why? Rip Rowan explained it best on prorec.com:

> Record labels have never really understood what makes a record sound good and frankly, few even care....For some reason, record labels have it in their heads that LOUD equals good, and therefore, LOUDER equals better. Not caring to understand even the basics of audio, these morons simply demand more volume (typically from the mastering engineer) and really don't understand or care about the consequences of their demands.

So using the audio excellence of Linux to rip major-label CDs is as pointless as burning copies of Xenix or SCO OpenServer. Record your own stuff. Freedom-loving hackers have reinvented the way we get operating systems and made them better. Now you can be the one to reinvent the way people get music, and make music better.

Two of the biggest advantages of 2.6 are going to be the included ALSA sound drivers and some dramatic improvements in latency that will make out-of-the-box distributions suitable for studio-quality audio. Preemptible kernel hacker Robert Love covers these and more on page 52.

If your need for raw software speed is greater than your attachment to little features like hardware memory protection, or if your embedded application is so important that if it goes down the machine might as well have crashed, you're the kind of person who might want to try Kernel Mode Linux. Toshiyuki Maeda explains how to make any application into a part of the kernel on page 62.

Hans Reiser follows up with more on the Reiser4 filesystem, one of several that are new for 2.6, on page 68. And, you might notice a few changes when you configure and build the new kernel. Greg Kroah-Hartman fills us in on what to expect on page 62.

Free software will always be an excellent choice for security and network applications, and a good example is the beginning of Mick Bauer's tutorial on Firewall Builder on page 30. The kernel is only a beginning. Freedom works. Use it.

Don Marti is editor in chief of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Adaptability and Ingenuity

Heather Mead

Issue #109, May 2003

Sometimes being the can-do person is a good thing, and sometimes it's not—advice for when the project rests on your shoulders.

Currently, my web article inbox is full of articles that reflect one of the tenets of the open-source philosophy—doing it yourself. Sometimes, however, we aren't doing things ourselves because we want to, but because it's our job and someone else waited too long to do his or hers. Other times, we are forced to find some way to pull it all together or watch as the whole process grinds to a halt. Out of necessity comes ingenuity, and that's most likely to be true if you're adaptable, which is one of the reasons why knowledge of Linux can be such a handy tool in your arsenal. From the articles I've seen lately, it seems that if you know Linux and open source, people are coming to you for help whether you want them to or not.

In "Installing Slash for a Private Project" (www.linuxjournal.com/article/6674), Paul Barry shares his tale of finding a way for an academic department to schedule meetings, decide on topics and record opinions and responses. To meet the department's asynchronous needs, Paul chose to use the Slashdot framework, Slash, installed locally. His article walks readers through the setup process, including installing the database back end, Apache with mod_perl support and all their dependencies. As always, Paul is humble enough to share his mistakes so you can avoid making the same ones.

If you're a system administrator, one of the most frustrating parts of your job may be dealing with people who expect you to reinvent the wheel on a day's notice. In "Configuring a Virtual Server Instance for Quick Recovery" (www.linuxjournal.com/article/6531), Jeffrey McDonald explains how he took advantage of VMware's disk modes, in both the server and workstations, to provide a new development/test environment in a day and a half. As he says, "It's cool to be able to run multiple instances of virtual servers on a single Linux host server, but easily being able to manage or back out changes to the OS and applications within the virtual server instances is even cooler.”

On the other end of the do-it-yourself spectrum, in a place we might call fun, is music—specifically, the theremin. Almost everyone wants to play some sort of instrument well. I'm still mad at my five-year-old self for refusing piano lessons. But the theremin offers us all a chance to make music in one of the most unique ways imaginable. To make it even easier, Seth David Schoen offers the “Poor Man's Theremin” (www.linuxjournal.com/article/6597), which “turns a laptop computer with an 802.11b card into a theremin-like instrument, using the signal strength reported by the card to control the pitch of a note”. Your coworkers and friends may complain, but at least they'll leave you alone for a while.

The weather is starting to break here in Seattle; we've already had a couple of those clear sunny days that exist to let you know there is more than work in life. So while we encourage everyone to take a break from the screen and keyboard—get crazy and go outside—we thank you for sharing all your stories and wait to see what you are up to next. Keep us posted at info@linuxjournal.com.

Heather Mead is senior editor of *Linux Journal*.

Best of Technical Support

Various

Issue #109, May 2003

Our experts answer your technical questions.

More Help for SSH Question

The first Best of Tech question in the March 2003 issue is a question that is becoming more and more common, because people and distributions are choosing higher security as a default or as an option. A possible reason that the user can't connect by SSH is the /etc/hosts.allow and /etc/hosts.deny files. Set **sshd: ALL** in hosts.allow, or preferably, if you know where you will SSH from, list only those hosts.

—Benjamin Judson

Partition Table Changes Don't Take

I am using a Seagate ST32550 SCSI hard drive with an AHA1720 interface card, but I am unable to partition it with fdisk. When I run fdisk on the drive, the changes do not become permanent, even after a reboot. The SCSI interface can detect it and can do low-level formats and verifications without a problem. When I enter **fdisk**, though, it creates the partition, but it does not stay put.

—Eskinder Mesfin, mesfin@attbi.com

It sounds like you are not writing the changes to the partition table; fdisk doesn't write until you tell it to write. Before you **q** to quit, do a **w** to write the changes.

—Christopher Wingert, cwingert@qualcomm.com

Connecting with MSN

My modem is configured to work with the KInternet program that is activated through the KDE desktop on SuSE 8.0. The modem initializes fine, calls the server of my ISP (MSN) and then it dies. I look at the activity log and see these error messages:

```
Failed Authentication with peer
Possible Bad Account or Bad Password
```

Does MSN require a different login process than what is accommodated under KInternet?

—Chris, cgsnip@msn.com

You might want to try a different authentication scheme, such as PAP or CHAP.

—Christopher Wingert, cwingert@qualcomm.com

Some users on mailing lists report success by prepending MSN/ to the user name. So if your user name were joe, you would set the user name in KInternet to **MSN/joe**.

—Don Marti, info@linuxjournal.com

Five Years without a Problem, Now This?

My SuSE 6.0 system has worked like a charm for almost five years nonstop—except for power failures—holding my DNS and Sendmail. Suddenly, the user account I always use is no longer allowed to log in. The only users that can log in are root and a second user account, but I can't figure why that user account is special. Although there's no login, I can **su** to any account by giving the correct password. The accounts are not locked; the passwords have not expired, the passwords are correct; the users have permissions on their home directories, and the permissions on passwd and shadow are correct. I've tried creating an account in the same group ID (admin) and groups as where the special account is listed—the one that can log in—but it didn't work. The messages in syslog are **incorrect password**.

—Juan Alvarez, juan.alvarez@thales-is.com

Without a closer look at your system, the gut reaction to this type of situation is to investigate the possibility of a system intrusion. Telnet sends your password in the clear over the network, and other dæmons installed on any five-year-old distribution have had vulnerability reports over the past few years. Your problem report does have that fishy smell. Barring any problems on that end, you can investigate some configuration facilities that control user logins. For example, is there an /etc/nologin file? This prevents any non-root user from logging in, and your extra user account may be given special treatment here if it is a member of the root group in /etc/group. Also, examine /etc/passwd and verify that the other users all have valid shells and home directories.

—Chad Robinson, crobinson@rfgonline.com

Given the age of the installation, you may want to upgrade to a newer and more secure distribution. A second guess would be the amount of available disk space.

—Christopher Wingert, cwingert@qualcomm.com

The two measures that prevent most security problems are 1) remove or disable unused software, which should include telnet—use OpenSSH and 2) subscribe to your distribution's security mailing list to get news of updates, then install the updates when they're available.

—Don Marti, info@linuxjournal.com

New Dell Server Locks Up

At work we are about to deploy our first web site that will run under Linux, which I'm quite happy about. However, I'm having a problem with the servers and hope you can help. Our development servers are Dell 2550 machines, and our production servers are Dell 2650s. We are running Red Hat 8.0 on the equipment, which runs fine for the most part. We have had unexplained lockups, however, on all the servers, in which the console becomes locked and the machine has to be hard reset. No indication of what caused the lockup is reported in any of the log files. After searching the Dell and Red Hat forums I've found some help. Essentially, this help is to put the option noapic on the kernel command line in the grub.conf file. After doing this, the machines seem to run well. What does the noapic command option do on an SMP system? And has anyone else experienced this problem on Dell 2550/2650 machines?

—Doug Farrell, dfarrell@grolier.com

The advanced programmable interrupt controller (APIC) replaces the standard, external interrupt controller with functionality inside the CPU itself. It supports some neat tricks such as performance counters and watchdog facilities. Normally, this support is not supposed to interfere with systems that do not have an APIC. However, in some instances this creates system lockups such as the ones you've experienced. The major implication of running in noapic mode is a performance hit, as interrupts are not handled as efficiently. For systems that are heavily interrupt-driven (this unfortunately includes those that do a lot of networking work, such as web servers) this might be measurable. Nonetheless, the benefits of SMP almost always outweigh this impact. Some load testing on your end should help you identify the maximum user loads that you can expect from your systems.

—Chad Robinson, crobinson@rfgonline.com

Belkin Wireless Card: Supported?

I am trying to get the Belkin wireless PCMCIA card to connect to a wireless access point from my laptop. I am wondering what module I should use for the PCMCIA card.

—Charles R. Fuller, charlesrfuller@netscape.net

Another Linux user was kind enough to post the details of his own experience with Belkin's wireless components on his web site. This site may be helpful to you: www.jacked-in.org/linux/belkin_wireless.php.

—Chad Robinson, crobinson@rfgonline.com

The Belkin card uses the same chipset as the Orinoco card. A simple solution is to alias the wireless device's Ethernet interface to **orinoco_cs** in /etc/modules.conf. If this does not work, you can find out more about the chipset with **cardctl ident**.

—Christopher Wingert, cwingert@qualcomm.com

Can't Boot New Install

When I attempt to boot Red Hat 7.3, I receive a message stating there is an invalid system disk. It also indicates I should replace the disk and press any key. What can I do to eliminate this problem short of reinstalling Linux?

—Logan, crossl@lakecitycc.edu

This message typically indicates that your BIOS was unable to find a boot loader on your drive. If you installed LILO or another boot manager when you installed Linux, chances are it was not properly done, and you should double-check the parameters you used. If you didn't install a boot manager, your problem is a bit easier to identify. Either way, you should be able to use an emergency recovery disk or the original installation disk to boot your system. Then you can install the boot loader again.

—Chad Robinson, crobinson@rfgonline.com

If there is a floppy in the disk drive, remove it.

—Christopher Wingert, cwingert@qualcomm.com

How to Make XDM Come Up at Boot?

I cannot get Linux (Red Hat 7.2) to boot into the X GUI. Instead, I get a login prompt. Is there a way to edit the default init level? It also fails WINE on boot up.

—Keith Raposo, keith.raposo@sms.siemens.com

To make sure X is correctly installed, type **startx**. If this works, you then can change the first non-comment line of /etc/inittab to

```
   id:5:initdefault:
```

—Usman S. Ansari, uansari@yahoo.com

In your /etc/inittab there is a line that reads **id:NUM:initdefault:**. Change the number to your desired init level, which is 5 for an XDM login screen.

—Christopher Wingert, cwingert@qualcomm.com

Archive Index Issue Table of Contents

Advanced search

New Products

Heather Mead

Issue #109, May 2003

Mobile DiskOnChip G3, Quicknet VoIP for Linux, SNAP Ultimate I/O Learning Center, and more.

Mobile DiskOnChip G3

M-Systems announced the release of the 64MB Mobile DiskOnChip G3, in collaboration with Toshiba, to meet the needs of feature-rich mobile devices, such as PDAs and 2.5G and 3G wireless devices. The G3 Flash disk is based on multilevel cell (MLC) NAND Flash memory, which reduces silicon size up to 50% by storing two bits per cell rather than one. The G3 boosts MLC NAND performance levels to those of binary NAND Flash rates. The G3 is available in a 7 × 10 × 1.2mm Fine-Pitch Ball Grid Array (FBGA) package and in a TSOP-I form factor for other connected and embedded devices.

Contact M-Systems, Inc., 8371 Central Avenue, Suite A, Newark, California 94560, 510-494-2090, info@m-sys.com, www.m-sys.com.

Quicknet VoIP for Linux

Quicknet Technologies announced Linux Special Edition products that can be used in concert with GnomeMeeting and Quicknet's MicroTelco VoIP services to make and receive voice calls over the Internet with a standard telephone set. Included in the Special Edition product line are internet PhoneJACK-PCI, LineJACK-ISA and PhoneCARD-PCMCIA add-in cards for quality voice transmission. Open-source drivers for the cards are included in the kernel, which combine with Quicknet's VoIP services, OpenH323 protocols and GnomeMeeting to allow low-cost, internet-based PC-to-phone calls worldwide.

Contact Quicknet Technologies, Inc., 520 Townsend Street, San Francisco, California 94103, 415-864-5225, www.quicknet.net.

SNAP Ultimate I/O Learning Center

The SNAP Ultimate I/O Learning Center is a standalone system that enables users to learn and train with Opto 22's SNAP Ultimate I/O system. The Learning Center includes a SNAP Ultimate processor, assorted I/O modules, a SNAP rack, power supply, load panel and cables. Also included are training manuals, a user's guide and a number of software applications and utilities to help users develop real-time industrial automation and capture-and-deliver applications. The Learning Center provides hands-on experience configuring I/O points, writing simple control strategies and building a graphical user interface.

Contact Opto 22, Inc., 43044 Business Park Drive, Temecula, California 92590, 800-321-OPTO, www.opto22.com.

GSX Server 2.5

The latest version of VMware's GSX Server, version 2.5, is now available. The enterprise-level virtual machine software is designed for business-critical applications in data centers and other high-traffic needs. GSX Server offers a secure and uniform platform for consolidating and partitioning servers to increase resource utilization and management efficiency. It can run multiple OSes and associated applications concurrently on a single Intel-based system. New features for version 2.5 include support for up to 64GB of host memory, up to 32 host processors and up to 64 active virtual machines.

Contact VMware, Inc., 3145 Porter Drive, Palo Alto, California 94304, 877-486-9273, sales@vmware.com, www.vmware.com.

S3C2500-RGP

Arcturus Networks and Samsung Electronics have partnered on a reference system, the S3C2500-RGP, for designers of residential gateways (RG), SOHO networks, internet attached devices (IAD) and convergence equipment. The system includes base hardware with extensible modules, the Linux OS and optional firmware suites to improve product functionality while speeding up design time. The platform is built on Samsung's ARM940 S3C2500 processor and is powered by µClinux. The system has 4MB of Flash ROM, 8MB of SDRAM, one 100BaseT Ethernet port, four 100BaseT Ethernet LAN switches, an on-chip cryptographic accelerator, two serial ports, PCMCIA support and I2C serial EPROM. Optional support is available for WiFi, WiFi/WAP, multiport DSP voice and SmartCard VPN authenticaion.

Contact Arcturus Networks, Inc., 116 Spadina Avenue, Suite 100, Toronto, Ontario M5V 2K6, Canada, 416-621-0125, info@arcturusnetworks.com, www.arcturusnetworks.com.

Repartee LX

The Repartee LX unified messaging system operates on Red Hat and offers users access and address communications from a wired or wireless telephone or from a networked PC. Repartee LX enables users to manage real-time telephone calls as well as voice mail and e-mail messages visually from their desktops. Text-to-speech capabilities for mobile users are provided by the ScanSoft RealSpeak speech engine. Features of Repartee LX include 16 international language prompts, analog and serial integrations, web-based system administration and support for 2-16 ports.

Contact ActiveVoice, Inc., 2033 Sixth Avenue, Suite 500, Seattle, Washington 98121, 800-284-3575, sales@activevoice.com, www.activevoice.com.

FAXCOM Server on Linux

FAXCOM Server on Linux features support for multiple diverse document attachments using on-the-fly document conversion, and it can support up to 96 ports on one fax server. Expanded fax-routing destination options for inbound faxes include fax port, dialed digits, sender's transmitting station identifier (TSID) and caller ID. When necessary, the same fax can be routed to multiple destinations, including one or more printers. FAXCOM Server on Linux includes a spam Fax Filter, which monitors incoming fax traffic and either eliminates spam traffic or quarantines it in a dedicated folder. FAXCOM Server on Linux also includes a fax-batching mechanism that enables users to send multiple faxes to the same telephone number in one call.

Contact Biscom, Inc., 321 Billerica Road, Chelmsford, Massachusetts 01824, 800-477-2472, sales@biscom.com, www.biscom.com.