Sun Technical Bulletin

August 1992

Technical Information Services

Sun Microsystems Computer Corporation
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 812-9208-01
Issue 1992 - 08

The *Sun Technical Bulletin* is distributed to customers with software/hardware or software-only support contracts. Send comments or corrections by electronic mail to *stb-editor@Sun.COM*, or by mail to:

Sun Technical Bulletin
Sun Microsystems Computer Corporation
2550 Garcia Avenue , M/S MPK02-04
Mountain View, CA 94043-1100 U.S.A.

Contents

STB Editor's Introduction 1

NOTICE — As of the July 1992 issue, the *Software Technical Bulletin* (STB) has changed its name to the "*Sun Technical Bulletin*". This is a change that we have been anticipating for some time, due to the fact that the STB occasionally covers hardware issues for the interest of our software customers.

This name change does not signify any change in the general purpose of the STB, nor in its audience, nor in its contractual obligations. The STB is still primarily a software document, delivered to customers with software (or software-and-hardware) support contracts. The Bulletin still covers primarily software-related topics, but with periodic coverage of hardware issues that may be of concern to software users—documentation problems, new commands and capabilities, performance benchmarks, and so on.

Thank you for your continued interest in...the *Sun Technical Bulletin*.

STB Overview

This month's STB contains articles of general interest in the topics outlined below.

Please note the *Updates* section at the end of the chapter. This section features new or updated information about articles from past issues of the STB. We appreciate the participation of our readership in calling this information to our attention.

Multithreading

"Beyond Multiprocessing – Multithreading the SunOS Kernel" is a technical paper from the SunSoft team that met the challenge of incorporating multithreading technology into the new kernel of SunOS 5.0 for the Solaris 2.0 distribution. This paper was written for the Summer 1992 USENIX meeting in San Antonio, Texas.

System Administration

This month, we begin a series of articles on System Administration in Solaris 2.0, Sun's operating environment based on System V Release 4 (SVR4). This material is extracted from Sun internal training materials.

The opening article, "System Administration in Solaris 2.0," offers a brief overview of the changes and other features available to the System Administrator in the new environment. "Directory Layout Changes" describes the revised file system structure of the principal directories; and "System Parameters" details parameters that either are new or can now be controlled dynamically without rebuilding the kernel.

In later issues, we will cover the Service Access Facility, Print Services, Boot and Run Levels, and other relevant topics.

Graphics

"Improving Performance with Direct Xlib" introduces the newly released 1.0 version of Direct Xlib, Sun's new, unbundled product that transparently improves the rendering performance of graphics-intensive Xlib-based applications.

Languages

We continue our series of articles relating to conversion of C programs for Solaris 2.0. "Dynamic Linking" is the preferred compilation method for System V Release 4 (SVR4). This article provides an overview of dynamic linking concepts and describes the improvements made in dynamic linking between SunOS 4.1.*x* and SVR4.

Tools

"SunSolve" introduces the newly shipped SunSolve product, an online service and support utility that delivers the latest and most important system information on CD-ROM to support contract customers, including common symptoms and resolutions, software bug reports, bug and patch descriptions, current and back issues of the *Customer Distributed Bugs List* and *Software Technical Bulletin*, various manuals, guides, and technical support articles, and more.

Software Release Versions

This section includes the current Sun software products and release version tables. We also include a revised updating of Sun's support matrix for Operating System release levels and platforms.

Notes & Comments

World Hotlines

For Sun customers world-wide served by your local service groups, use the customer service telephone numbers listed in this monthly item. Also, look to this section during the upcoming year for details on your local support call policies and procedures.

Reporting Bugs

The information in this monthly section continues to be expanded as Sun software service centers are added world-wide. The section includes a list of Sun service centers, addresses, e-mail hotlines, telephone hotlines, and FAX numbers.

Your STB Subscription

This notice is published monthly, giving customers useful information regarding ordering multiple STB hardcopy subscriptions, and duplicating additional STB copies from a single "master" copy. This duplication permission is limited, as detailed in the note.

Updates

This section presents updates on subjects covered in past issues of the Software Technical Bulletin.

SVR4 vs. BSD Printing (May 1992)

This article included a section on “Setting Up Remote Printers” under Solaris 2.0, which contained an error in one given command:

2. Set up the remote printer:

```
# lpsystem -t bsd printserver_name
# lpadmin -p printer_name -s printserver_name   ⇐ ERROR
# /usr/lib/lpsched
# accept printer_name
# enable printer_name
# lpadmin -d printer_name
```

The first `lpadmin` command should read:

```
# lpadmin -p printer_name -s printserver_name\!printer_name
```

We regret any confusion that this error may have caused.

Changing OpenWindows Default Font Size (May 1992)

This article, while discussing the method of changing OpenWindows fonts, mentioned in passing the “new DVT035 display terminal…a compact 13" monitor…”. This product was, in fact, not named DVT035 upon its release in April 1992, nor was it a 13" monitor: the product was the new Mid-Range 16" monitor that is an option with the SPARCstation IPC (order number 4/40EC-8-P40).

Availability of Perl (December 1991, April 1992)

We remind readers who have expressed unfamiliarity with the Perl language that the July issue reprinted excerpts from a common information file on that language. The entire file is available to readers upon request.

Further information can be obtained from Larry Wall, the author of the language, at e-mail address *lwall@netlabs.COM*.

We would like to encourage our readership to e-mail us any comments or suggestions regarding the content of the Sun Technical Bulletin, including requests for articles on specific subject matters. We are aiming at a broad spectrum of topics that are of interest to all sectors of our audience, both highly experienced technical people and relatively inexperienced users who are new to desktop workstations. We are not the appropriate channel for researching any and all technical problems that may come up in the field, but we will do our best to follow up on questions about articles that we have printed in the recent past.

Please let us know what you think, and we will do our best to accommodate you in future issues.

Thanks.

The STB Editor
sun!stb-editor or *stb-editor@Sun.COM*

Multithreading 2

Beyond Multiprocessing – Multithreading the SunOS Kernel

By – J.R. Eykholt, S.R. Kleiman, S. Barton, R. Faulkner, A. Shivalingiah, M. Smith, D. Stein, J. Voll, M. Weeks, D. Williams – SunSoft, Inc.

The following article is a SunSoft technical paper presented at USENIX, Summer 1992, San Antonio, Texas.

Introduction

Preparing the SunOS/SVR4 kernel for today's challenges—symmetric multiprocessing, multithreaded applications, real-time, and multimedia—led to the incorporation of several innovative techniques. In particular, the kernel was restructured around threads. Threads are used for most asynchronous processing, including interrupts. The resulting kernel is fully preemptible and capable of real-time response. The combination provides a robust base for highly concurrent, responsive operation.

When we started to investigate enhancements to the SunOS kernel to support multiprocessors, we realized that we wanted to go further than merely adding locks to the kernel and keeping the user process model unchanged. It was important for the kernel to be capable of a high degree of concurrency on tightly coupled symmetric multiprocessors, but it was also a goal to support more than one thread of control within a user process. These threads must be capable of executing system calls and handling page faults independently. On multiprocessor systems, these threads of control must be capable of running concurrently on different processors. Powell's 1991 paper described the user-visible thread architecture.[1]

We also wanted the kernel to be capable of bounded dispatch latency for real-time threads.[2] Real-time response requires absolute control over scheduling, requiring preemption at almost any point in the kernel, and elimination of unbounded priority inversions wherever possible.

The kernel itself is a very complex multithreaded program. Threads can be used by user applications as a structuring technique to manage multiple asynchronous activities; the kernel benefits from a thread facility that is essentially the same.

The resulting SunOS 5.0 kernel, the central operating system component of Solaris 2.0, is fully preemptible, has real-time scheduling, symmetrically supports multiprocessors, and supports user-level multithreading. Several of the locking strategies used in this kernel were described by Kleiman et al.[3] In this paper we'll describe some of the implementation features that make this kernel unique.

Overview of the Kernel Architecture

A kernel thread is the fundamental entity that is scheduled and dispatched onto one of the CPUs of the system. A kernel thread is very lightweight, having only a small data structure and a stack. Switching between kernel threads does not require a change of virtual memory address space information, so it is relatively inexpensive. Kernel threads are fully preemptible and may be scheduled by any of the scheduling classes in the system, including the real-time (fixed priority) class. Since all other execution entities are built using kernel threads, they represent a fully preemptible, real-time "nucleus" within the kernel.

---

1. M.L. Powell, S.R. Kleiman, S. Barton, D. Shah, D. Stein, M. Weeks, *SunOS Multi-thread Architecture*, USENIX, Winter 1991, Dallas, TX.
2. Sandeep Khanna, Michael Sebrée, John Zolnowsky, *Realtime Scheduling in SunOS 5.0*, USENIX, Winter 1992, San Francisco, CA [hereafter referred to as "Khanna 1992"].
3. S. Kleiman, J. Voll, J. Eykholt, A. Shivalingiah, D. Williams, M. Smith, S. Barton, and G. Skinner, *Symmetric Multiprocessing in Solaris 2.0*, COMPCON, Spring 1992, p. 181, San Francisco, CA.

Kernel threads use synchronization primitives that support protocols for preventing priority inversion, so a thread's priority is determined by which activities it is impeding by holding locks as well as by the service it is performing.[4]

SunOS uses kernel threads to provide asynchronous kernel activity, such as asynchronous writes to disk, servicing STREAMS queues, and callouts. This removes various diversions in the idle loop and trap code and replaces them with independently scheduled threads. Not only does this increase potential concurrency (these activities can be handled by other CPUs), but it also gives each asynchronous activity a priority so that it can be appropriately scheduled.

Even interrupts are handled by kernel threads. The kernel synchronizes with interrupt handlers via normal thread synchronization primitives. If an interrupt thread encounters a locked synchronization variable, it blocks and allows the critical section to clear.

A major feature of the new kernel is its support of multiple kernel-supported threads of control, called lightweight processes (LWPs), in any user process, sharing the address space of the process and other resources, such as open files. The kernel supports the execution of user LWPs by associating a kernel thread with each LWP, as shown in Figure 2-1. While all LWPs have a kernel thread, not all kernel threads have an LWP.

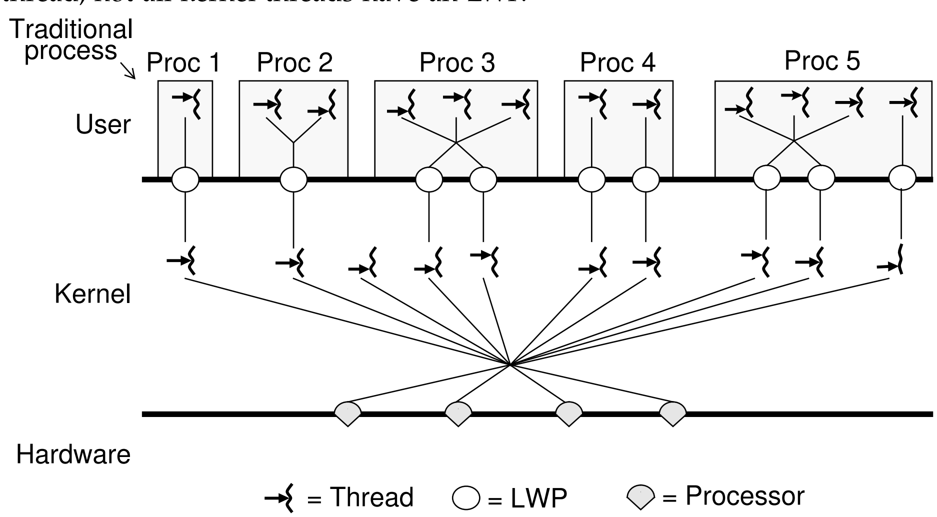

Figure 2-1 Multithread Architecture Examples

4. Khanna 1992.

A user-level library uses LWPs to implement user-level threads.[5] These threads are scheduled at user-level and switched by the library to any of the LWPs belonging to the process. User threads can also be bound to a particular LWP. Separating user-level threads from the LWP allows the user thread library to quickly switch between user threads without entering the kernel. In addition, it allows a user process to have thousands of threads, without overwhelming kernel resources.

Data Structures

In the traditional kernel, the `user` and `proc` structures contained all kernel data for the process. Processor data was held in global variables and data structures. The per-process data was divided between non-swappable data in the `proc` structure, and swappable data in the `user` structure. The kernel stack of the process, which is also swappable, was allocated with the `user` structure in the user area, usually one or two pages long.

The restructured kernel must separate this data into data associated with each LWP and its kernel thread, the data associated with each process, and the data associated with each processor. Figure 2-2 shows the relationship of these data structures in the restructured kernel.

5. D. Stein, D. Shah, *Implementing Lightweight Threads*, USENIX, Summer 1992, San Antonio, TX.

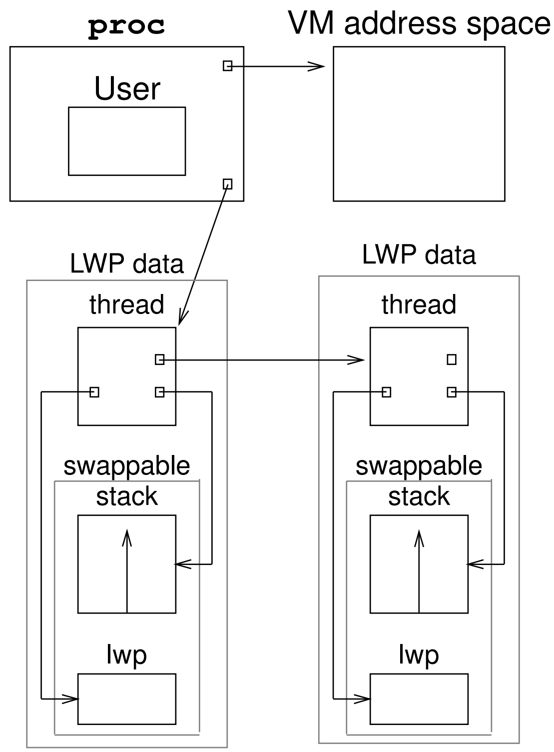

Figure 2-2 Multithread Data Structures for a Process

The per-process data is contained in the `proc` structure. It contains a list of kernel threads associated with the process, a pointer to the process address space, user credentials, and the list of signal handlers. The `proc` structure also contains the vestigial `user` structure, which is now much smaller than a page, and is no longer practical to swap.

The LWP structure contains the per-LWP data such as the process-control-block (`pcb`) for storing user-level processor registers, system call arguments, signal handling masks, resource usage information, and profiling pointers. It also contains pointers to the associated kernel thread and process structures. The kernel stack of the thread is allocated with the LWP structure inside a swappable area.

The kernel thread structure contains the kernel registers, scheduling class, dispatch queue links, and pointers to the stack and the associated LWP, process, and CPU structures. The thread structure is not swapped, so it also contains some data associated with the LWP that is needed even when the LWP structure is swapped out. Thread structures are linked on a list of threads for the process, and also on a list of all existing threads in the system.

Per-processor data is kept in the `cpu` structure, which has pointers to the currently executing thread, the idle thread for that CPU, and current dispatching and interrupt handling information. There is a substructure of the `cpu` structure that can be architecture-dependent, but the main body is intended to be applicable to most multiprocessing architectures.

To speed access to the thread, LWP, process, and CPU structures, the SPARC implementation uses a global register, `%g7`, to point to the current thread structure. A C-preprocessor macro, `curthread`, allows access to fields in the current thread structure with a single instruction. The current LWP, process, and CPU structures are quickly accessible through pointers in the thread structure. In the future we may dedicate additional global registers for other frequently accessed structures.

Kernel Thread Scheduling

SunOS 5.0 provides several scheduling classes. A scheduling class determines the relative priority of processes within the class, and converts that priority to a global priority. With the addition of multithreading, the scheduling classes and dispatcher operate on threads instead of processes. The scheduling classes currently supported are system, timesharing, and real-time (fixed-priority).

The dispatcher chooses the thread with the greatest global priority to run on the CPU. If more than one thread has the same priority, they are dispatched in round-robin order.

The kernel has been made preemptible to better support the real-time class and interrupt threads. Preemption is disabled only in a small number of bounded sections of code. This means that a runnable thread runs as soon as is practical after its priority becomes high enough. For example, when thread A releases a lock on which higher priority thread B is sleeping, the running thread A immediately puts itself back on the run queue and allows the CPU to run thread B. On a multiprocessor, if thread A has better priority than thread B, but thread B has better priority than the current thread on another CPU, that CPU is directed to preempt its current thread and choose the best thread to run. In addition, user code run by an underlying kernel thread of sufficient priority (e.g., real-time threads) will execute even though other lower priority kernel threads wait for execution resources.[6]

System Threads

System threads can be created for short or long-term activities. They are scheduled like any other thread, but usually belong to the system scheduling class. These threads have no need for LWP structures, so the thread structure and stack for these threads can be allocated together in a non-swappable area, as shown in Figure 2-3.

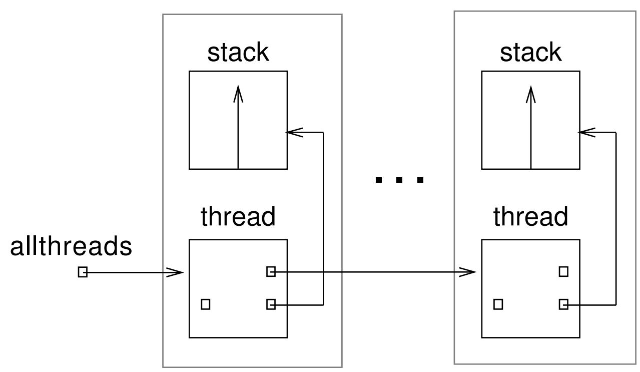

Figure 2-3 System Threads

A new segment driver, `seg_kp`, handles stack allocations. It handles virtual memory allocations for the kernel that can be paged or swapped out; it also provides "red zones" to protect against stack overflow. System threads use `seg_kp` for the stack and the thread structure, in a non-swappable region. LWPs use it to allocate the LWP structure and kernel stack in a swappable region.

6. Further details can be found in Khanna 1992.

Synchronization Architecture

The kernel implements the same synchronization objects for internal use as are provided by the user-level libraries for use in multithreaded application programs.[7] These are mutual exclusion locks (*mutexes*), condition variables, semaphores, and "multiple readers, single writer" (readers/writer) locks. The interfaces are shown below.[8]

Mutual exclusion locks

```
void   mutex_enter(kmutex_t *lp);
void   mutex_exit(kmutex_t *lp);
void   mutex_init(kmutex_t *lp, char *name,
          kmutex_type_t type, void *arg);
void   mutex_destroy(kmutex_t *lp);
int    mutex_tryenter(kmutex_t *lp);
```

Condition variables

```
void   cv_wait(kcondvar_t *cp, kmutex_t *lp);
int    cv_wait_sig(kcondvar_t *cp, kmutex_t *lp);
int    cv_timedwait(kcondvar_t *cvp, kmutex_t *lp, long tim);
void   cv_signal(kcondvar_t *cp);
void   cv_broadcast(kcondvar_t *cp);
```

Multiple readers, single writer locks

```
void   rw_init(krwlock_t *lp, char *name, krw_type_t type,
          void *arg);
void   rw_destroy(krwlock_t *lp);
void   rw_enter(krwlock_t *lp, krw_t rw);
int    rw_tryenter(krwlock_t *lp, krw_t rw);
void   rw_exit(krwlock_t *lp);
void   rw_downgrade(krwlock_t *lp);
int    rw_tryupgrade(krwlock_t *lp);
```

7. M.L. Powell, S.R. Kleiman, S. Barton, D. Shah, D. Stein, M. Weeks, *SunOS Multi-thread Architecture*, USENIX, Winter 1991, Dallas, TX.

8. Note that kernel synchronization primitives must use a different type name than user synchronization primitives so that the types are not confused in applications that read internal kernel data structures.

Counting semaphores

```
void   sema_init(ksema_t *sp, unsigned int val, char *name,
   ksema_type_t type, void *arg);
void   sema_destroy(ksema_t *sp);
void   sema_p(ksema_t *sp);
int    sema_p_sig(ksema_t *sp);
int    sema_tryp(ksema_t *sp);
void   sema_v(ksema_t *sp);
```

These are all implemented such that the behavior of the synchronization object is specified when it is initialized. Synchronization operations, such as acquiring a mutex lock, take a pointer to the object as an argument and may behave somewhat differently depending on the type and optional type-specific argument specified when the object was initialized.

Most of the synchronization objects have types that enable collecting statistics such as blocking counts or times. A patchable kernel variable can also set the default types to enable statistics gathering. This allows the selection of statistics gathering on particular synchronization objects or on the kernel as a whole.

The semantics of most of the synchronization primitives cause the calling thread to be prevented from progressing past the primitive until some condition is satisfied. The way in which further progress is impeded (e.g., sleep, spin, or other) is a function of the initialization. By default, the kernel thread synchronization primitives that can logically block, can *potentially* sleep.

Some of the synchronization primitives are strictly bracketing (e.g., the thread that *locks* a mutex must be the thread that *unlocks* it) and a single owner can be determined (i.e., mutexes and writer locks). In these cases, the synchronization primitives support the priority inheritance protocol, as described by Khanna et al.[9]

Some synchronization primitives are intended for situations where they may block for long or indeterminate periods. Variants of some of the primitives are provided (e.g., `cv_wait_sig()` and `sema_p_sig()`) that allow blocking to be interrupted by a reception of a signal. There is no non-local jump to the head of the system call, as there was in the traditional `sleep` routine. When a

9. Khanna 1992.

signal is pending, the primitive returns with a value indicating the blocking was interrupted by a signal and the caller must release any resources and return.

Mutual Exclusion Lock Implementation

Mutual exclusion locks (mutexes) prevent more than one thread from proceeding when the lock is acquired. They prevent races on access to shared data and are by far the most heavily used primitive.

Mutexes are usually held for short intervals. For example, it would not be good to hold a critical system mutex while waiting for disk I/O to complete. Mutexes are not recursive; the owner of the lock cannot again call `mutex_enter()` for the same lock. If a thread holds a mutex, the same thread must be the one to release the mutex. These rules are enforced to promote good programming practice and to avoid deadlocks.

If `mutex_enter` cannot set the lock (because it is already set), the blocking action taken depends on the mutex type that was passed to `mutex_init`, and stored in the mutex. The default blocking policy for mutexes, called *adaptive* (type `MUTEX_DEFAULT`), spins while the owner of the lock (recorded when the lock is acquired) remains running on a processor. This is done by polling the owner's status in the spin wait loop.[10] If the owner ceases to run, the caller stops spinning and sleeps.[11] This gives fast response and low overhead for simple contention.

Spin mutexes are available as type `MUTEX_SPIN`, which takes as its type-specific argument the interrupt level to be disabled while the mutex is held. It is rarely used, as adaptive mutexes are more efficient, in general.

Device drivers are restricted to using type `MUTEX_DRIVER`, which takes a Sun-DDI-defined opaque value as an argument. This argument is basically an interrupt priority in the current implementation, and determines whether the blocking policy is adaptive or spin, based on whether the interrupt priority is above the "thread level" (see below).

10. In order to avoid locking while inspecting the owner's status during the spin, the state is determined indirectly. The algorithm spins while the current thread pointer of any CPU points to the owning thread, indicating it is running.

11. On uniprocessors, this turns into always sleeping, since the owner cannot be running.

A simple trick speeds up `mutex_enter()` for adaptive mutexes. Non-adaptive mutexes use a separate primitive lock field in the mutex data structure, with the lock field used by the adaptive type always in the locked state. This is so that `mutex_enter()` can always attempt to apply an adaptive lock first, and only if that fails, consider the possibility that the mutex might be another type.

Turnstiles vs. Queues in Synchronization Objects

Each synchronization object requires a way of finding threads that are suspended waiting for that object. It is important to keep the storage cost of synchronization objects small, because many system structures contain synchronization objects, so the queue header is not directly in the object. Instead, two bytes in the synchronization object are used to find a *turnstile* structure containing the sleep queue header and priority inheritance information.[12] Turnstiles are preallocated such that there are always more turnstiles than the number of threads active.

One alternative method would be to select the sleep queue from an array using a hash function on the address of the synchronization object. This is essentially the approach used by `sleep()` in the traditional kernel. The turnstile approach is favored for more predictable real-time behavior, since they are never shared by other locks, as hashed sleep queues sometimes are.

Interrupts as Threads

Many implementations[13] [14] have a variety of synchronization primitives that have similar semantics (e.g., mutual exclusion) yet explicitly sleep or spin for blocking. For mutexes, the spin primitives must hold interrupt priority high enough while the lock is held to prevent any interrupt handlers that may also use the synchronization object from interrupting while the object is locked, causing deadlock. The interrupt level must be raised before the lock is acquired and then lowered after the lock is released.

12. Khanna 1992.

13. Graham Hamilton and Daniel S. Conde, *An Experimental Symmetric Multiprocessor Ultrix Kernel*, USENIX, Winter 1988, Dallas, TX.

14. Kent Peacock, Sunil Saxena, Dean Thomas, Fred Yang and Wilfred Yu, *Experiences from Multithreading System V Release 4*, Symposium on Experiences with Distributed & Multiprocessor Systems (SEDMS) III, March 1992, Newport Beach, CA.

This has several drawbacks. First, the raising and lowering of interrupt priority can be an expensive operation, especially on architectures that require external interrupt controllers (remember that mutexes are heavily used). Secondly, in a modular kernel, such as SunOS, many subsystems are interdependent. In several cases (e.g., mapping in kernel memory or memory allocation) these requests can come from interrupt handlers and can involve many kernel subsystems. This in turn, means that the mutexes used in many kernel subsystems must protect themselves at a relatively high priority from the *possibility* that they may be required by an interrupt handler. This tends to keep interrupt priority high for relatively long periods and the cost of raising and lowering interrupt priority must be paid for every mutex acquisition and release. Lastly, interrupt handlers must live in a constrained environment that avoids any use of kernel functions that can potentially sleep, even for short periods.

To avoid these drawbacks, the SunOS 5.0 kernel treats most interrupts as asynchronously created and dispatched high-priority threads. This enables these interrupt handlers to sleep, if required, and to use the standard synchronization primitives.

On most architectures putting threads to sleep must be done in software. This must be protected from interrupts if interrupts are to sleep themselves or wake up other threads. The restructured kernel uses a primitive spin lock protected by raised priority to implement this. This is one of a few bounded sections of code where interrupts are locked out.

Traditional UNIX kernel implementations[15] [16] also protect the dispatcher by locking out interrupts, usually all interrupts. The restructured kernel has a modifiable level (the "thread level") above which interrupts are no longer handled as threads and are treated more like non-portable "firmware" (e.g., simulating DMA via programmed I/O). These interrupt handlers can only synchronize using the spin variants of mutex locks and software interrupts. If the "thread level" is set to the maximum priority, then all interrupts are locked out during dispatching. For implementations where the "firmware" cannot tolerate even the relatively small dispatcher lockout time, the "thread level" can be lowered. Typically this is lowered to the interrupt level at which the scheduling clock runs.

---

15. Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, John S. Quarterman, *The Design and Implementation of the 4.3 BSD UNIX Operating System*, Addison-Wesley, Reading, MA, 1989.

16. Maurice J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.

Implementing Interrupts as Threads

Previous versions of SunOS have treated interrupts in the traditional UNIX way. When an interrupt occurs the interrupted process is held captive (*pinned*) until the interrupt returns. Typically, interrupts are handled on the kernel stack of the interrupted process or on a separate interrupt stack. The interrupt handler must complete execution and get off the stack before anything else is allowed to run on that processor. In these systems the kernel synchronizes with interrupt handlers by blocking out interrupts while in critical sections.

In SunOS 5.0, interrupts behave like asynchronously created threads. Interrupts must be efficient, so a full thread creation for each interrupt is impractical. Instead, we preallocate interrupt threads, already partly initialized. When an interrupt occurs, we do the minimum amount of work to move onto the stack of an interrupt thread, and set it as the current thread. At this point, the interrupt thread and the interrupted thread are not completely separated. The interrupt thread is not yet a full-fledged thread (it cannot be descheduled) and the interrupted thread is *pinned* until the interrupt thread returns or blocks, and cannot proceed on another CPU. When the interrupt returns, we restore the state of the interrupted thread and return.

Interrupts may nest. An interrupt thread may itself be interrupted and be pinned by another interrupt thread.

If an interrupt thread blocks on a synchronization variable (e.g., mutex or condition variable), it saves state (*passivates*) to make it a full-fledged thread, capable of being run by any CPU, and then returns to the pinned thread. Thus most of the overhead of creating a full thread is only done when the interrupt must block, due to contention.[17]

While an interrupt thread is in progress, the interrupt level it is handling, and all lower-priority interrupts, must be blocked. This is handled by the normal interrupt priority mechanism unless the thread blocks. If it blocks, these interrupts must remain disabled in case the interrupt handler is not reenterable at the point that it blocked or it is still doing high-priority processing (i.e., should not be interrupted by lower-priority work). While it is blocked, the interrupt thread is bound to the processor it started on as an implementation convenience and to guarantee that there will always be an interrupt thread

17. On SPARC this overhead involves flushing the entire register file. This is only done if the interrupt handler sleeps, not during interrupt handling without contention.

available when an interrupt occurs (though this may change in the future). A flag is set in the `cpu` structure indicating that an interrupt at that level has blocked, and the minimum interrupt level is noted. Whenever the interrupt level changes, the CPU's base interrupt level is checked, and the actual interrupt priority level is never allowed to be below that.

There is also an interface which allows an interrupt thread to continue as a normal, high-priority thread. When `release_interrupt()` is called, it saves the state of the pinned thread and clears the indication that the interrupt thread has blocked, allowing the CPU to lower the interrupt priority level.

An alternative approach to this is to use bounded first-level interrupt handlers to capture device state and then wake up an interrupt thread that is waiting to do the remainder of the servicing.[18] This approach has the disadvantages of requiring device drivers to be restructured and of always requiring a full context switch to the second level thread. The approach used in SunOS 5.0 allows full thread behavior without restructured drivers and with very little additional cost in the no-contention case.

Interrupt Thread Cost

The additional overhead in taking an interrupt is about 40 SPARC instructions. The savings in the mutex enter/exit path is about 12 instructions. However, mutex operations are much more frequent than interrupts, so there is a net gain in time cost, as long as interrupts don't block too frequently. The work to convert an interrupt into a "real" thread is performed only when there is lock contention.

There is a cost in terms of memory usage also. Currently an interrupt thread is preallocated for each potentially active interrupt level below the thread level for each CPU.[19] An additional interrupt thread is preallocated for the clock (one per system). Since each thread requires a stack and a data structure, perhaps 8K bytes or so, the memory cost can be high.

18. David Barnett, *Kernel Threads and their Performance Benefits*, Real Time, Vol. 4, No. 1, Lynx Real-Time Systems, Inc., Los Gatos, CA., 1992.

19. There are nine interrupt levels on the Sun SPARC implementation that can potentially use threads.

However, it is unlikely that all interrupt levels are active at any one time, so it is possible to have a smaller pool of interrupt threads on each CPU and block all subsequent interrupts below the thread level when the pool is empty, essentially limiting how many interrupts may be simultaneously active.

Clock Interrupt

The clock interrupt[20] is handled specially. There is only one clock interrupt thread in the system (not one per CPU), and the clock interrupt handler invokes the clock thread only if it is not already active.

The clock thread could possibly be delayed for more than one clock tick by blocking on a mutex or by higher-level interrupts. When a clock tick occurs and the clock thread is already active, the interrupt is cleared and a counter is incremented. If the clock thread finds the counter non-zero before it returns, it will decrement the counter and repeat the clock processing. This occurs very rarely in practice. When it occurs, it is usually due to heavy activity at higher interrupt levels. It can also occur while debugging.

Kernel Locking Strategy

The locking approach used almost exclusively in the kernel to ensure data consistency is data-based locking. That is, the mutex and readers/writer locks each protect a set of shared data, as opposed to protecting routines (monitors). Every piece of shared data is protected by a synchronization object.

Some aspects of locking in the virtual memory, file system, STREAMS, and device drivers have already been discussed by Kleiman et al.[21] Here we'll elaborate a bit on device driver issues, as they are closely related to interrupt threads.

Non-MT Driver Support

Some drivers haven't been modified to protect themselves against concurrency in a multithreaded environment. These drivers are called *MT-unsafe*, because they don't provide their own locking.

---

20. This occurs 100 times a second on current Sun SPARC implementations.

21. S. Kleiman, J. Voll, J. Eykholt, A. Shivalingiah, D. Williams, M. Smith, S. Barton, and G. Skinner, *Symmetric Multiprocessing in Solaris 2.0*, COMPCON, Spring 1992, p. 181, San Francisco, CA.

In order to provide some interim support for *MT-unsafe* drivers, we provided wrappers that acquire a single global mutex, `unsafe_driver`. These wrappers insure that only one such driver will be active at any one time. This wrapper is illustrated by Figure 2-4.

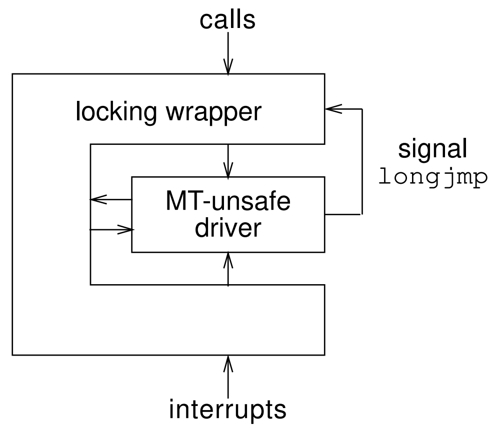

Figure 2-4 Unsafe Driver Wrapper

There are several ways a driver may be entered, from the explicit driver entry points, interrupts, and callbacks. Each of these entries must acquire the `unsafe_driver` mutex if the driver isn't safe. For example, if an MT-unsafe driver uses `timeout()` to request a function call at a later time, the `callout` structure is marked so that the `unsafe_driver` mutex will be held during the function call.

MT-unsafe drivers can also use the old `sleep/wakeup` mechanism. `sleep()` safely releases the `unsafe_driver` mutex after the thread is asleep, and reacquires it before returning.

The `longjmp()` feature of `sleep()` is maintained as well. When a thread is signalled in `sleep()`, if it specified a dispatch value greater than `PZERO`, a `longjmp()` takes the thread to a `setjmp()` that was performed in the unsafe driver entry wrapper, which returns `EINTR` to the caller of the driver.

`sleep()` checks to make sure it is called by an MT-unsafe driver, and panics if it isn't. It isn't safe to use `sleep()` from a driver which does its own locking.

It is fairly easy to provide at least simple locking for a driver, so almost all drivers in the system have some of their own locking. These drivers are called *MT-safe*, regardless of how fine-grained their locking is. Some developers have used the term *MT-hot* to indicate that a driver does fine-grained locking.

SVR4/MP DKI Locking Primitives

As we implemented our driver interfaces, UNIX International and USL were defining the SVR4 Multiprocessor Device Driver Interface and Driver-Kernel Interface (DDI/DKI), with a different set of locking primitives, based around the traditional UNIX interrupt-blocking model.

SunOS 5.0 implements those interfaces to the extent defined so far, using our locking primitives and ignoring any spin semantics. This allows drivers using those interfaces to be more easily ported. SunOS drivers typically use the SunOS synchronization primitives.

Implementation Technology

Some interesting techniques made it easier to get this all working.

Kernel Time Slicing

Since the kernel is fully preemptible we were able to make kernel threads time-slice. We simply added code to the clock interrupt handler to preempt whatever thread was interrupted. This allows even a uniprocessor to have almost arbitrary code interleavings. Increasing the clock interrupt rate made this even more valuable in finding windows where data was unprotected. By causing kernel threads to preempt each other as often as possible we were able to find locking problems using uniprocessor hardware before multiprocessor hardware was available. Even when working multiprocessor hardware arrived, there were far more uniprocessors available than multiprocessors. We intend this only as a debugging feature, since it does have some adverse performance impact, however slight.

Lock Hierarchy Violation Detection

Instead of establishing a system lock hierarchy *a priori*, we developed a static analysis tool that would check for lock ordering violations in the system. This lint-like tool, called *locknest*, reads C source code, constructs call graphs and reports on locking cycles. We feel it helped during early implementation debugging, and probably reduced the amount of time spent debugging deadlocks. A similar tool is described by Korty.[22]

Deadlock Detection

A side-benefit of the priority inheritance mechanism[23] is that deadlocks caused by hierarchy violations are usually detected at run time as well. It does a good job on mutexes and readers/writer locks held for write, but since there isn't a complete list of threads holding a read lock, it can't always find deadlocks involving readers/writer locks. There are other deadlocks possible with condition variables; these aren't detected.

Summary

SunOS 5.0 is a multithreaded and symmetric multiprocessor version of the SVR4 kernel. The primary features are:

- Fully preemptible, real-time kernel
- High degree of concurrency on symmetric multiprocessors
- Support for user threads
- Interrupts handled as independent threads
- Adaptive mutual-exclusion locks

The thread models inside the kernel and at user level are almost identical. The scheduling of kernel threads onto CPUs is analogous to the way the threads library schedules user-level threads onto LWPs. The use of threads for structuring the kernel has mostly good effects though they can be overused. Threads do have a cost. The stacks are large, and must be allocated on separate pages if protection for potential stack overrun is needed. Also, context switching is still expensive. Some things are still better implemented by callouts and other "zero-weight" processes, but threads provide a nice structuring paradigm for the kernel.

---

22. Joe Korty, *Sema: a Lint-Like Tool for Analyzing Semaphore Usage in a Multithreaded Unix Kernel*, USENIX, Winter 1989, San Diego, CA.

23. Khanna 1992.

Principal References

Sandeep Khanna, Michael Sebrée, John Zolnowsky, *Realtime Scheduling in SunOS 5.0*, USENIX, Winter 1992, San Francisco, California.

This describes the real-time features and considerations in this kernel.

S. Kleiman, J. Voll, J. Eykholt, A. Shivalingiah, D. Williams, M. Smith, S. Barton, and G. Skinner, *Symmetric Multiprocessing in Solaris 2.0*, COMPCON, Spring 1992, p. 181, San Francisco, California.

M.L. Powell, S.R. Kleiman, S. Barton, D. Shah, D. Stein, M. Weeks, *SunOS Multi-thread Architecture*, USENIX, Winter 1991, Dallas, Texas.

This describes the architecture for user-level multithreading.

D. Stein, D. Shah, *Implementing Lightweight Threads*, USENIX, Summer 1992, San Antonio, Texas.

This describes the implementation of the user-level threads package.

Author Information

Joseph Eykholt is a Senior Staff Engineer and technical leader in the OS-Multithreading group at SunSoft. He received an MSEE from Purdue University in 1978, and a BSEE from Purdue in 1977. Prior to coming to Sun, he was one of the leading developers of multiprocessing features for the Amdahl UTS system, and a logic designer for the Amdahl 580 CPU.

Steve Kleiman is a Distinguished Engineer in the Operating Systems Technology Department of SunSoft. He is currently architect of Multithreading in SunOS. He received an MS in Electrical Engineering from Stanford University in 1978 and a BS in Electrical Engineering and Computer Science from MIT in 1977. He has been involved with the design and development of UNIX and workstation architecture since 1977, first at Bell Telephone Laboratories and then at Sun. He was one of the developers of NFS, Vnodes, and the original port of SunOS to SPARC.

Steve Barton graduated from the University of California, Santa Cruz in 1982 with a BA in Computer and Information Sciences. Since then he has worked at Zilog Inc., Parallel Computers, CounterPoint Computers, and Telestream Corporation. He is currently a Member of Technical Staff at SunSoft. He has been with Sun for the last four years.

Roger Faulkner is a Senior Staff Engineer in the OS-Multithreading group at SunSoft. He received a BS in Physics from North Carolina State University in 1963 and a PhD in Physics from Princeton University in 1967, then joined Bell Laboratories, where he was seduced by computers. He has been actively involved in the inner workings of the UNIX kernel since 1976 and has done compiler and debugger development along the way. He is one of the principals involved in the development of the `/proc` file system for SVR4.

Anil Shivalingiah is a Staff Engineer in the OS-Virtual Memory group at SunSoft. He received an MS in Computer Science from University of Texas, Arlington in 1983 and a BS in Electronics Engineering from UVCE, India in 1981. He has been with Sun for the last three years.

Mark Smith graduated with a BS in Computer Science from the University of California, Santa Barbara in 1986. He is currently a Member of Technical Staff at SunSoft in the OS-Multithreading group. Prior to coming to Sun, he worked in the Design Automation department of Amdahl Corp.

Dan Stein is currently a Member of Technical Staff at SunSoft where he is one of the developers of the SunOS Multi-thread Architecture. He graduated from the University of Wisconsin in 1981 with a BS in Computer Science.

Jim Voll works in the OS-Multithreading group at SunSoft. He received his BS from the University of California, Santa Barbara in 1981. Prior to working at Sun, he has worked at Amdahl and Cygnet Systems.

Mary Weeks has been a member of technical staff at Sun Microsystem since 1986. Prior to Sun, she worked at Xerox. She received her BA in computer science from the University of California at Berkeley in 1984.

Dock Williams is a Staff Engineer in the OS-Multithreading group at SunSoft. He received an SB [sic] in Electrical Engineering and Computer Science from MIT in 1980. He has been with Sun for over six years. Prior to joining Sun, he worked at American Information Systems, ONYX Systems, Tri-Comp Systems, and Hughes Aircraft Radar Systems.

Solaris 2.0 3

System Administration in Solaris 2.0

This month, we begin a series of articles on System Administration in Solaris 2.0, Sun's operating environment based on System V Release 4 (SVR4).

Industry Standard Merged UNIX

In general, the concept used to merge the historical UNIX variants into SVR4 and the Solaris 2.0 operating environment has been to consolidate the existing functionality while maintaining compatibility for existing applications. As a result, features and commands tended to be added to the product, with few features being withdrawn.

SVR4 is the consolidation of the three major UNIX strains:

- System V Release 3
- XENIX
- BSD/SunOS

A small number of features of SunOS 4.1.*x* were not merged into the base SVR4. Existing users of SunOS 4.1.*x* are still enjoying those features, and existing applications may rely on them, so the excluded features were packaged together and included in the Solaris 2.0 operating environment in the form of *compatibility packages*.

Throughout this chapter, transitional diagrams will be used to illustrate the features in Solaris 2.0 as compared with the contents of SunOS 4.1.*x*. The diagrams will show which features of SunOS 4.1.*x* are being withdrawn and which are being retained, and which features are new to Solaris 2.0. Figure 3-1 demonstrates the transitional diagram model:

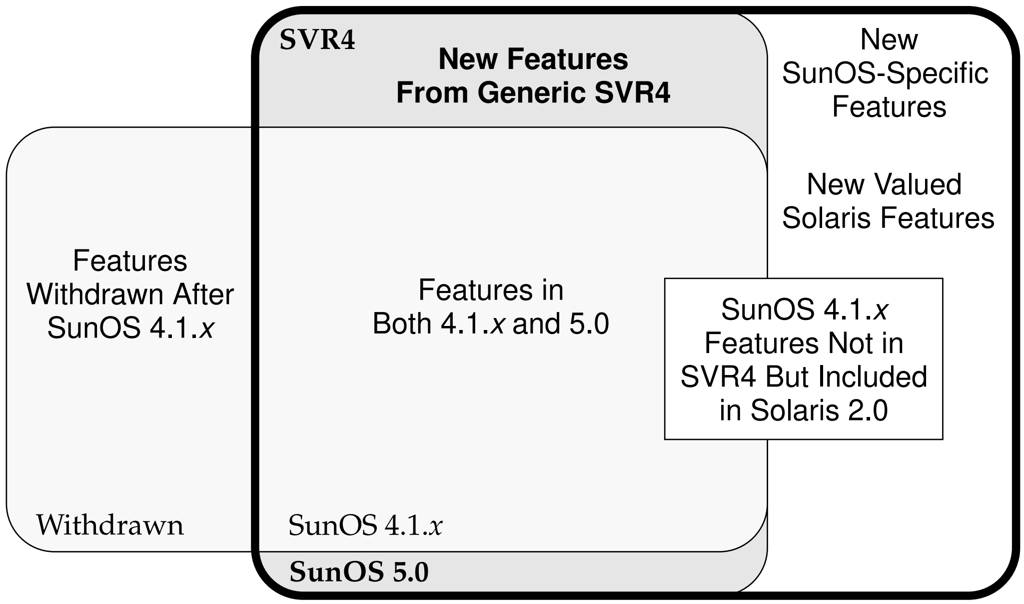

Figure 3-1 Transitional Diagram Model

SVR4 Contents for Systems Administrators

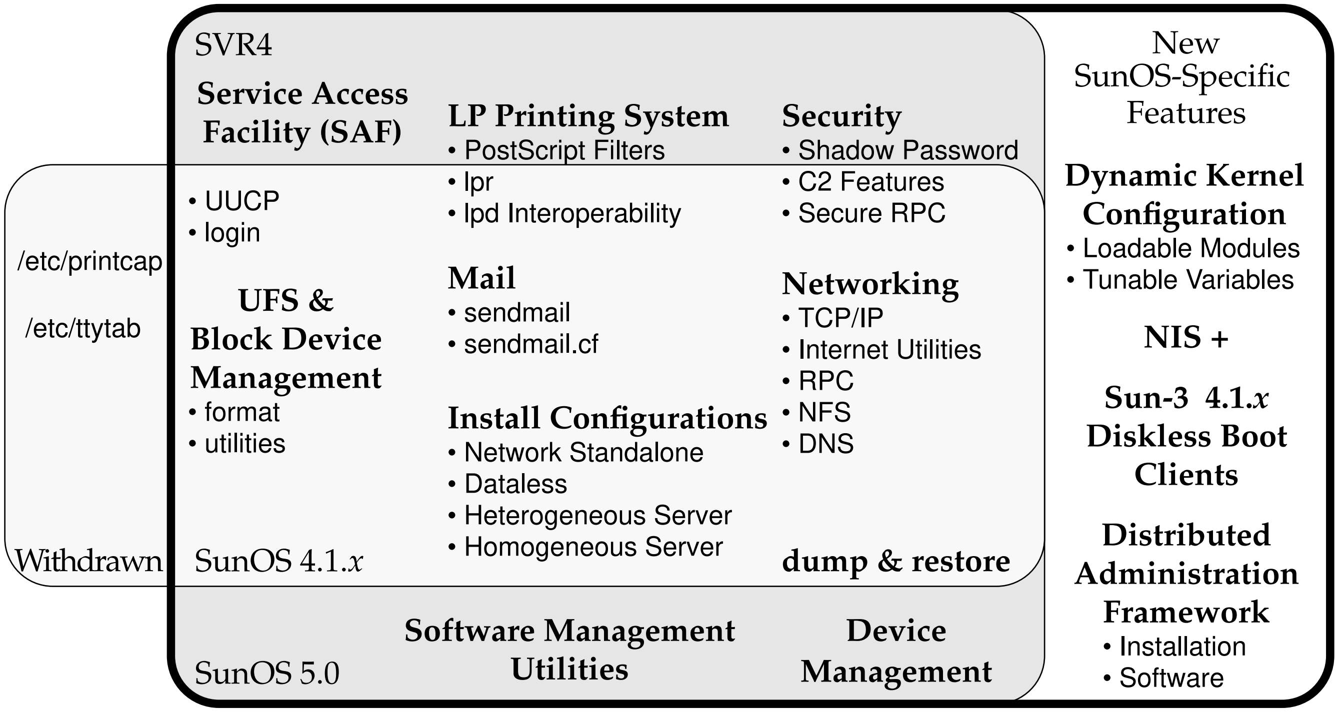

Figure 3-2 SVR4 Contents for System Administrators

Service Access Facility

The Service Access Facility (SAF) provides a consistent method on administration for connection oriented services compared to previous releases. It controls serial port access services such as login and `getty` service.

LP Printing System

The SVR4 LP Printing system provides new features traditionally found on more expensive computing systems. The system comes with bundled PostScript filters for laser printers.

Security

Security is tightened on SVR4 systems, with improved features such as more sophisticated password aging controls, restricted shell access controls, and a shadow password file designed to foil unauthorized intruders.

Device Management Utilities

SVR4 provides optional new utilities that simplify the administration of large systems with many peripheral devices. The device management utilities enable the orderly allocation of devices among processes. Similar devices can be classified into pools, and requests for devices from programs are queued in an orderly fashion.

Software Management Utilities

SVR4 provides new utilities to simplify the maintenance of software by users and system administrators. Utilities are included to add and remove software, and to display the software currently installed. Management of software will be easier because system software, Sun layered software products, and third-party applications will all be manageable in the same way.

The Solaris 2.0 Operating Environment Features

Dynamically Configurable Kernel

Administrators will never again have to spend time configuring kernels, because the Solaris 2.0 operating environment is based upon a dynamically configurable kernel. Users can add devices by using a simple command. All Solaris 2.0 drivers are packaged as loadable modules for simplicity. Administrators can adjust kernel integer variables by resetting a field in a system file, rather than rebuilding the kernel.

NIS+ Name Service

The NIS+ Name Service, an enhanced version of NIS+, provides an improved security model for maps, simpler administration through the use of a hierarchical name space that can be administered by different groups, and faster updates via a mechanism that propagates changes only--not entire maps.

Sun-3 4.1.x Diskless Boot Client Support

To assist customers in the transition to the Solaris 2.0 operating environment, diskless boot clients running SunOS 4.1.*x* releases will be supported on Solaris 2.0 servers.

Distributed Administration Framework

Future releases of Solaris will allow flexible distributed system administration using a simple graphical user interface. The Solaris 2.0 operating environment provides the underlying technology for this flexible framework, and supplies GUI tools to manage system configuration and software.

Directory Layout Changes

The Solaris 2.0 directory structure is very similar to the SunOS 4.1.2 directory structure. The major difference is that important system directories and files are stored in slightly different locations or their names have changed. For example:

- The `rc` scripts used to change system states are stored in `/sbin`, and are linked to the `/etc` directory. The actual scripts used to start and stop system services are stored in `/etc/init.d` .
- The kernel (now called `unix`) and all kernel-level object modules are stored in the `/kernel` directory. The `/sys` directory is no longer needed because of a dynamically loaded kernel.
- Two new directories have been added to the directory structure:
  - The optional `/opt` file system can be used to store third-party or unbundled software. If `/opt` is not a separate file system, it is a symbolic link to `/usr/opt`.
  - The `/proc` file system can be used to access all active processes on the system.

The main goal for these changes is to maintain a consistent purpose for directories and accommodate changes to SVR4 in an orderly fashion.

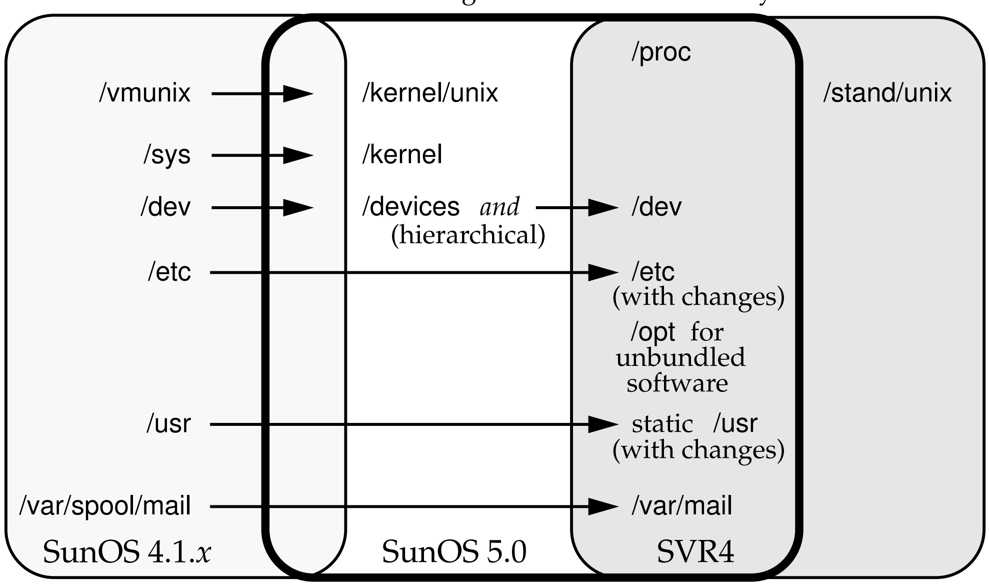

Figure 3-3 Overview of Changes to Directory Layout

The `root` File System

The /(root) file system is very similar to the previous SunOS release. However, there are some minor changes.

The kernel is now called `unix`, and it is stored in the `/kernel` directory, which also contains the loadable kernel modules.

The SunOS 4.*x* `/sys` directory is obsolete. Now that the kernel is dynamically loaded, there is no need for a `/sys` directory holding kernel source and unlinked object files.

The commands in `/sbin` are still used to mount the `/usr` directory, and it also includes the `rc` scripts that are used to change system states.

The new directories and executables include:

`/devices/*`

Records the state of physically connected devices. Devices in the `/dev` directory are symbolically linked to the new device names in this directory, using OBP device naming.

`/kernel/*`

This directory contains the kernel (`unix`) and loadable kernel modules.

`/opt/*`

This file system (or directory) may contain unbundled software and third-party products.

`/proc/*`

This directory contains a file representing an image of each active process.

`/sbin/*`

Although `/sbin` still included the commands necessary to mount `/usr`, it also contains the `rc` scripts used to change system run levels.

`/stand/*`

This directory is currently empty. It contains standalone boot files in generic SVR4.

`/ufsboot`

This is the Solaris 2.0 boot program.

The /etc Directory

The /etc directory contains machine-specific information and system administration commands, like the previous SunOS release. It also contains many new administration commands.

The new files and directories include:

/etc/RELEASE
: This file contains the latest operating system installation information.

/etc/TIMEZONE
: The timezone variable (TZ) in this file is initially set during installation.

/etc/cron.d/*
: This directory contains files for controlling and monitoring cron activities.

/etc/default/*
: This directory contains files of default settings controlling the behavior of the su, passwd, login, and other utilities.

/etc/fs/*
: This directory contains file system-specific executables needed before /usr is mounted, such as mount.

/etc/iu.ap
: This is a new configuration file of the STREAMS modules to be pushed for the console device [*not normally adminstrated*].

/etc/inet/*
: this directory contains Internet-related files such as /etc/hosts, /etc/networks, /etc/protocols, and /etc/services, which are symbolic links to this directory.

/etc/init.d/*
: This directory contains scripts used to start and stop system services.

/etc/inittab
: This file dictates the definition of run levels to the init process.

`/etc/lp/*`
This directory contains configuration and information files for the printing service. Information formerly found in `/etc/printcap` is now in this directory.

`/etc/mail/*`
This directory contains electronic mail administration files, such as `aliases` and `sendmail.cf` .

`/etc/mnttab`
This file contains the mounted file system table.

`/etc/net/*`
This directory contains control files used by transport-independent networking [*not normally adminstrated*].

`/etc/netconfig`
A new network configuration file used with the transport-independent RPC networking. Each entry defines a transport available on the system.

`/etc/nodename`
The host name is set by this file.

`/etc/nsswitch.conf`
This new file describes the name service sources to be used for each type of service lookup, such as “hosts”.

`/etc/rc*`
The `rc` scripts in `/sbin` are symbolically linked to `/etc` for compatibility purposes.

`/etc/rc*.d/*`
The scripts in `/etc/init.d` used to start and stop system services are linked to these directories.

`/etc/saf/*`
This directory contains configuration information used by the Service Access Controller to define port monitors and services.

`/etc/shadow`
An entry for each user and encrypted password is contained in this file.

`/etc/system`

This new file is used to override kernel specifications such as the default search path for modules, the default root and swap devices, and to force the loading of modules at boot time.

`/etc/telinit`

This command is used to change system run levels and is linked to `/sbin/init`. It replaces the SunOS booting and halting commands.

`/etc/tm/*`

This directory can be used to hold trademark files, which are displayed at boot time, and is referenced from the `/etc/init.d/sysetup` script.

`/etc/ttydefs` *and* `/etc/ttysrch`

The `ttydefs` file replaces `gettytab` as the terminal line settings database. The `ttysrch` file [*not normally adminstrated*] can be used to speed the search for `tty` devices in the `/dev` directory.

`/etc/vfstab`

This file contains default values for mounting file systems and remote resources. It replaces `fstab` from the previous SunOS release.

`/etc/utmpx` *and* `/etc/wtmpx`

These files contain user and accounting information for the `login`, `who`, and `write` commands, and are linked to the `/var/adm` directory. `umtpx` is an extended version of `utmp`.

The `/var` *Directory*

As in the previous SunOS release, the `/var` directory contains machine- and user-specific data such as log files and mail and spooling files. In addition, `/var` also contains information about installed software packages.

The `/var` directory is similar to the previous SunOS release with some additional directories:

`/var/cron/*`

This directory contains a `cron` log file.

`/var/lp/*`

This directory contains `lp` log files.

/var/mail/*

This directory contains mailbox files (it replaces /var/spool/mail).

/var/news/*

This directory can contain news files.

/var/nis/*

This directory contains configuration and name data used by NIS+.

/var/opt/*

This directory is created and used by application packages.

/var/options/*

This directory contains files (symbolic links) to /var/sadm/pkg/*, which identify each utility installed on the system.

/var/sadm/*

This directory contains configuration and log files for software installation and package management utilities. For example, the file /var/sadm/install/contents lists everything installed on the system.

/var/saf/*

This directory contains Service Access Facility log files.

The /opt *File System*

The purpose of the /opt directory or file system is to store unbundled or third-party software. Having a separate file system for software helps keep the /usr file system static. The following software categories are common examples of what may optionally be loaded into /opt :

SUNWste

This directory contains the executables and files for SPARCWorks 2.0, which includes components for SPARC components, C and C++ language tools:

SW2.0/bin	This directory contains the binaries.
SW2.0/lib	This directory contains the configuration files, help files, icons and libraries.
SW2.0/man	This directory contains the manual pages.

sundiag

This directory contains the executables for SunDiag 4.0, the first OpenWindows-based version of the diagnostics tool.

The /proc File System

The /proc file system provides access to images of active processes via standard system calls (open, close, read, write, and ioctl). Each filename in /proc identifies a process ID, which represents currently running processes on the system.

The /proc file system is automatically mounted via an entry in the /etc/vfstab file. System administrators are not required to maintain /proc. Its main purpose is for debugging process images.

Each filename indicates the decimal number of the process ID it represents (e.g., /proc/00957, /proc/01019, and so on). Each filename is owned by the process's user ID.

The /usr File System

The purpose of the /usr file system is similar to the previous SunOS release in that it contains executable commands, system programs, and library routines.

With the addition of the /opt directory for unbundled and third-party software, /usr can remain fairly static.

New /usr Directories

The new directories in /usr depend on what software packages are installed. General new directories include:

/usr/4lib/*
: This directory contains the binary compatibility libraries.

/usr/aset/*
: This directory contains the files and directories used by the Automated Security Enhancement package.

/usr/ccs/*
: This directory (C compilation system) contains compiler support tools, as well as any bundled programming tools, such as make.

/usr/options/*
: This directory contains package administration related files.

`/usr/sadm/*`
This directory contains *sysadm* related tools

`/usr/snadm/*`
This directory contains distributed administration executables.

`/usr/sbin/*`
This directory contains administration commands, including the `installboot` program.

`/usr/ucb/*`
This compatibility directory contains the SunOS/BSD executables formerly located in `/usr/bin` .

`/usr/ucbinclude/*`
This compatibility directory contains the SunOS/BSD include files formerly located in `/usr/include` .

`/usr/ucblib/*`
This compatibility directory contains the SunOS/BSD libraries formerly located in `/usr/lib` .

Changed `/usr` *Directories*

The following SunOS 4.*x* directories have changed:

`/usr/5bin`
Now a link to the native System V commands in `/usr/bin`.

`/usr/5include`
The directory is removed. The native System V include files are in `/usr/include`.

`/usr/5lib`
The directory is removed. The native System V libraries are in `/usr/lib`.

`/usr/etc`
The directory is removed. It formerly held miscellaneous file system-specific executables, which are better organized in Solaris 2.0.

`/usr/old`
This directory's contents are removed at each new major release.

/usr/xpg2bin
The directory is removed. Its former contents are now in /usr/bin.

/usr/xpg2lib
The directory is removed. Its former contents are now in /usr/lib.

System Parameters

The new dynamic kernel configuration in Solaris 2.0 has simplified the way in which kernel parameters are configured.

Kernel Parameters in SunOS 4.x

Most kernel parameters in SunOS 4.*x* indicated the size of static kernel tables which were allocated at boot time. The process of tuning these variables involved editing the kernel configuration file or `param.c`, followed by the compilation and installation of the newly created kernel. The final step would be booting the system. Note that the main drawbacks were the time-consuming procedure just described and the potential misuse of memory caused by the oversizing of these tables.

Kernel Parameters in Solaris 2.0

Kernel parameter configuration in Solaris 2.0 has changed in a number of ways. Given that the kernel is no longer compiled, there is no kernel configuration file in which to change the default values of kernel parameters. This is now achieved by creating entries in a new system configuration file named `/etc/system`.

Most of the tables that previously were statically allocated at boot time are now dynamically allocated — i.e., they will grow as needed up to a limit. This limit or *high-water mark* is the value of the parameter that controls the size of the table. Thus the parameters that used to indicate the size of tables to be allocated at boot time, now indicate the maximum size that these tables may reach.

Most kernel parameters have changed from SunOS 4.*x* . In some cases, their names were changed; in others, they disappeared completely. There are, however, a few which have retained both their names and functions. One such parameter is `maxusers`, which still performs the same function it did in SunOS 4.*x* — it controls the value of other system parameters such as:

`max_nprocs`	maximum number of processes
`ufs_ninode`	maximum number of in-core inodes
`max_uproc`	maximum number of processes a user can have

Other variables that can be tuned are:

`shminfo_shmmax`	maximum shared memory segment size
`tune_t_fsflushr`	rate at which `fsflush` is run (in seconds)
`rstchown`	is POSIX restricted `chown` or not?

A white paper explaining Solaris 2.0 kernel parameters in greater depth will be available in the near future.

The File `/etc/system`

The file `/etc/system` is the kernel parameter configuration file, superceding `param.c` and the kernel configuration file in SunOS 4.*x* . The new file contains directives which are read by the kernel at initialization time. These directives are used to manipulate kernel modules, root file system, and kernel parameters.

The file's contents resemble the following:

```
EXCLUDE: namespace/module
INCLUDE: namespace/module
FORCELOAD: namespace/module
ROOTDEV: root_device_name
ROOTFS: root_fs_type
SWAPDEV: swap_device_name
SWAPFS: swap_fs_type
MODDIR: path1:path2:...
SET [module:]symbol {=,|,&} [~][-] value
```

The available *namespace*s are:

`drv`	device drivers
`exec`	execution format modules
`fs`	filesystem modules
`sched`	scheduling modules
`strmod`	STREAMS modules
`sys`	loadable system call modules
`misc`	miscellaneous modules

Note that these namespaces coincide with the names of the various subdirectories in `/kernel`.

The available *directives* are:

EXCLUDE	Do not allow the named loadable module to be loaded
INCLUDE	Include the named loadable module (default)
FORCELOAD	Force the named module to be loaded during kernel initialization
ROOTDEV	Set the root device
ROOTFS	Set the root file system type
SWAPDEV	Set the swap file system
SWAPFS	Set the swap file system type
MODDIR	Set the search path for loadable kernel modules
SET	Set a kernel or module variable

Here are some examples of `/etc/system` directives:

- To force the loading of a module:
  ```
  FORCELOAD: sys/sysacct
  ```
- To prevent the loading of a module:
  ```
  EXCLUDE: sys/C2audit
  ```
- To set a kernel parameter:
  ```
  SET maxusers=4
  ```
- To set a parameter in a kernel module:
  ```
  SET shmsys:shminfo_shmmax=100000
  ```

STREAMS Networking Drivers

To configure the STREAMS-based networking drivers, use the utility `/usr/sbin/ndd`.

- Parameters of networking streams devices can be displayed and set with `/usr/sbin/ndd`
- To view the list of parameters supported by a driver:
  ```
  # ndd /dev/tcp \?
  ```
- To set a parameter such as `ip_forwarding`:
  ```
  # ndd -set /dev/ip ip_forwarding 0
  ```
- To view the `arp` cache:
  ```
  # ndd /dev/arp arp_cache_report
  ```

Graphics 4

Improving Performance with Direct Xlib

By – Jamshed Doctor, SMCC Marketing Development Engineering

Direct Xlib is Sun Microsystems Computer Corporation’s new, unbundled product that transparently improves the rendering performance of graphics-intensive Xlib-based applications. The improved performance depends on the application's use and mix of graphics primitives. Applications using the standard Xlib Application Programming Interface (API) should not have to make any changes to their source code, because Direct Xlib presents the exact same MIT X11R4 interface.

Direct Xlib is a set of supplemental (dynamic and static) Xlib libraries for OpenWindows 3.0. It can be easily disabled when standard server rendering is desired, and can be used for either the entire window system or on an application-by-application basis. A dynamically linked application can use Direct Xlib without any recompile or relink. A statically linked application must relink with the static Direct Xlib library provided. Performance increase is provided only when both the window client and the window server are on the same machine. Otherwise, standard Xlib is used for rendering.

Direct Xlib Implementation

Direct Xlib increases rendering performance by using Direct Graphics Access (DGA) technology to access the graphics hardware directly. It guarantees pixel accuracy of graphics output to be identical to that produced by OpenWindows by using the same rendering code as the X11/NeWS server. Most new code implemented for Direct Xlib is for synchronization between Direct Xlib and the server. Performance improvements come from eliminating the overhead involved in copying X11 commands and data into the X11 command buffer, eliminating the overhead of interprocess communication (IPC) between the client and the server in transferring this command buffer, and kernel process management overhead.

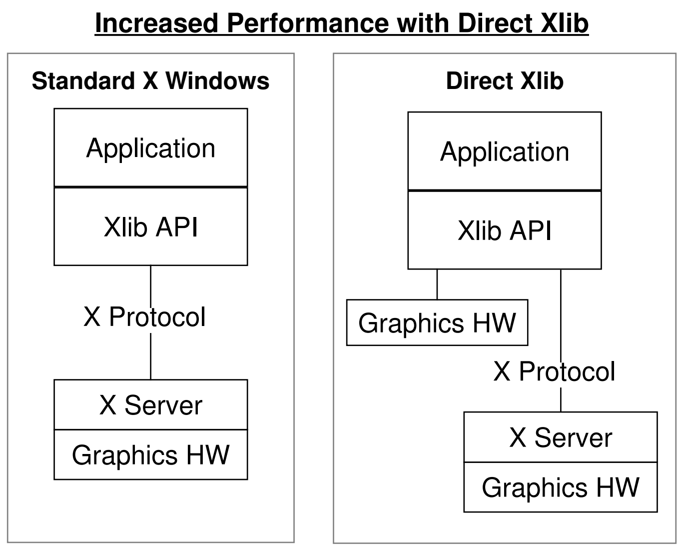

Figure 4-1 Increased Performance with Direct Xlib

Direct Xlib Performance Numbers

Direct Xlib roughly doubles raw graphics performance. An application can typically achieve a graphics performance improvement of approximately 50%, due to inefficiencies in its graphics code. Most window system operations do not achieve a noticeable performance increase, because they are limited by application software overhead rather than graphics software. Enhancing performance of large window scrolling is limited by the hardware blit performance in OpenWindows 3.0. The greatest performance increase is achieved for large numbers of short vectors, filled rectangles, short text strings, and small scrolls or blit operations.

Figure 4-2 shows the performance data from the "x11perf" benchmark, conducted by Direct Xlib engineering on a SPARCstation2 GX machine. It compares OpenWindows 3.0 X11/NeWS server with the Direct Xlib product.

Graphics Primitives	Direct Xlib	OpenWindows 3.0	Ratio
Dots	1,410,000	364,000	3.87
Lines	431,000	254,000	1.07
Rectangles	321,000	150,000	2.14
Text	195,000	104,000	1.88
Window Operations			
Scroll	12,300	3010	4.09
Copy	12,600	2910	4.33
Put Image	40.9	6.0	6.82

Figure 4-2 Direct Xlib Performance Data

- Lines are 10 pixel randomly oriented.
- Rectangles are 10x10 solid.
- Text is Times Roman 10, 80 character unclipped polytext line.
- Scroll and Copy are for 10x10 pixel areas.
- PutImage is for 500x500 pixel areas.

Direct Xlib speeds up graphics rendering, but not general computation. Programs that are not graphics-intensive may see little overall performance gain. Direct Xlib will provide better performance for programs that are limited by X graphics performance. If other factors are limiting the performance of the program, Direct Xlib offers only minimal improvements. Non-graphics portions of Xlib code are not accelerated by Direct Xlib.

Direct Xlib Supported Primitives

Direct Xlib accelerates the graphics primitives that are heavily used by graphics-intensive applications and can produce significant performance improvements when rendered directly by the client-side code. The rest of the primitives are sent to the server for graphics rendering. Even those primitives that are Direct Xlib-accelerated are sent to the server for rendering under certain conditions. Generally, these graphics primitives are accelerated (rendered locally) when Direct Xlib can be assured that there is no possibility that it will render them incorrectly or incompletely.

Direct Xlib accelerates the following primitives:

- `XCopyArea` – when Direct Xlib is sure that no graphics exposure events other than a `NoExpose` event need to be generated
- `XPutImage` – when bit ordering, byte ordering, scanline padding, etc., match the native attributes used internally within the server
- `XSetClipRectangles` – to support local clip control
- `XSetClipMask` – supported locally only when the client has local DGA access to the pixmap used as a clip mask (available with future versions of Direct Xlib on Solaris 2.0)
- `XDrawPoint` / `XDrawPoints`
- `XDrawline` / `XDrawLines` / `XDrawSegments`
- `XFillRectangle` / `XFillRectangles`
- `XDrawRectangle` / `XDrawRectangles` – For line width less than or equal to one and number of rectangles less than or equal to two, these commands are concerted to batched `XDrawLine` commands. For all other line widths or when more than two rectangles are requested, Direct Xlib uses the same significantly slower rendering techniques as the OpenWindows server.
- `XDrawArc` / `XDrawArcs` / `XFillArc` / `XFillArcs` – These primitives are rendered locally, when possible. `XDrawArcs` are not batched due to the way join styles are applied to arcs.This is also the case for standard Xlib.
- `XDrawPoint` / `XDrawPoints` – are batched locally in the client with delayed rendering to provide maximum performance, as in standard Xlib

- XDrawLine, XDrawPoint, XFillRectangle, XDrawRectangle, and XFillArc primitives – batched locally in the client with delayed rendering to provide maximum performance, as they are in standard Xlib.
- XFillPolygon – Polygons are batched locally in the client with delayed rendering to provide maximum performance. A total of 256 vertices among one or several polygons is batched locally.
- XDrawString / XDrawImageString – Direct Xlib currently supports a maximum of thirty-two (32) 8-bit fonts stored locally on the client side. The amount of local memory used to store font bitmaps can be controlled by the DIRECTX_FONT_MEMORY_LIMIT environment variable. The default value is 65536 bytes per client. Most 8-bit fonts average between 4K and 8K in total memory needed for bitmaps.
- XClearWindow / XClearArea – Only the simplest case of these primitives is supported locally: when the background is rendered using a single background pixel color and is not ParentRelative.

Direct Xlib also supports local DGA access to backing store.

Programming Tips: Improve Your Application Performance

How much an application's performance improves depends on the application's graphics utilization and mix of graphics primitives. Almost everything that applies to making standard Xlib graphics fast, applies to Direct Xlib; on Direct Xlib this performance gain is magnified. Optimizing an application's graphics code would include the following:

- batching vectors into chains
- avoiding color changes when possible
- avoiding copies of data within the application

Pixels for all drawables accelerated (rendered locally) by Direct Xlib are shared with OpenWindows. Access to these pixels is synchronized between the two processes. Synchronization is accomplished by a combination of the locking mechanism, and by flushing graphics primitives and forcing a round trip to the server through the X11 command buffer whenever necessary. To minimize round trips, synchronization with the server is implemented only when switching between client local and server-based graphics rendering. If a series of primitives can be rendered locally, no round trip-style synchronization is necessary. Conversely, if a series of primitives can only be rendered by the server, no extra round trip synchronization is necessary. Round trip-style synchronization is only required when alternating between local and server based graphics rendering.

Not all operations are faster with Direct Xlib. Applications that use many fill styles or clip masks may actually show performance reductions when running with Direct Xlib on OpenWindows 3.0. This is because DGA-shared pixmaps are not supported in OpenWindows 3.0; the amount of synchronization necessary with the server can increase significantly. This limitation is removed in the future version of Direct Xlib on Solaris 2.0. In addition, window creation and destruction times are significantly slower with Direct Xlib because of extra overhead involved in gaining direct access to the windows. Because the DGA interface forces all changes to the window hierarchy to complete immediately without the benefits of batching requests and delayed processing, the performance of operations like window circulation suffers when many windows exist in the hierarchy.

Direct Xlib and the X Window System

To get accelerated performance from the application, Direct Xlib must be used with OpenWindows 3.0 X11/NeWS server. If any other MIT X11 server is used, you will not see any acceleration because DGA will not be available on other X servers.

Using other X servers should not cause any noticeable degradation in performance in comparison to the standard Xlib libraries.

Positioning Direct Xlib, XGL, and Other Graphic APIs on SPARCstation

XGL offers maximum performance and functionality on a SPARCstation, and is the recommended API for all graphics intensive applications. Under Sun's Open Graphics Initiative, several XGL-based APIs are available—SunPHIGS, SunGKS, Nth Graphics' NPGL, Ithaca Software's HOOPS—all of which offer high performance and enhanced functionality in comparison to Direct Xlib.

For applications that must use the Xlib API due to portability considerations, or those that don't need XGL's advanced functionality, the use of Direct Xlib can provide a significant graphics performance gain.

Benefits from Direct Xlib for XGL API and Applications

There is currently no support for raster text in XGL 2.0 API. An XGL-based application that must use raster text may make an Xlib call to display raster text. Raster text is accelerated by Direct Xlib, and the application may see improved performance.

XGL also uses Xlib to render wide lines. If Direct Xlib is enabled, it will get called. Wide lines are supported by Direct Xlib, but are unlikely to be any faster than through standard Xlib.

In the case where XGL renders through Xlib (dumb frame buffers) because DGA is not available, Direct Xlib will not help, because it needs DGA as well. In this case, standard server rendering will be used.

Systems and Software Requirements for Using Direct Xlib

Direct Xlib is currently supported on the following platforms:

- Systems: Sun-4, Sun-4c, and Sun-4m.
- Operating System: SunOS 4.1.1 GFX Rev. 2, SunOS 4.1.2, and later versions.
- Window System: OpenWindows 3.0 and later versions.
- Graphics Accelerators: GX, GXplus, GS, and GT. Acts like standard Xlib for `bwtwo0`, `cgthree0`, and other dumb frame buffers supported by OpenWindows 3.0.

Installing Direct Xlib

Direct Xlib is a set of supplemental Xlib libraries for OpenWindows 3.0. Some key points to be aware of during installation:

- OpenWindows 3.0 should be installed before installing Direct Xlib.
- Installation must be performed as `root`.
- Direct Xlib must be installed on a local disk.
- At least 4 Megabytes of disk space should be made available for installation.
- Installation directory may be in the `$OPENWINHOME` directory, where OpenWindows 3.0 is installed. Direct Xlib can be installed in another location, if desired.

The supported system configuration during installation includes:

- Sun-4 Local Standalone Workstation
- Local Homogeneous Server Workstation
- Local Heterogeneous Server Workstation

The following figure illustrates the file system structure on the CD as it is shipped. The Direct Xlib installation directory is shown in Figure 4-3:

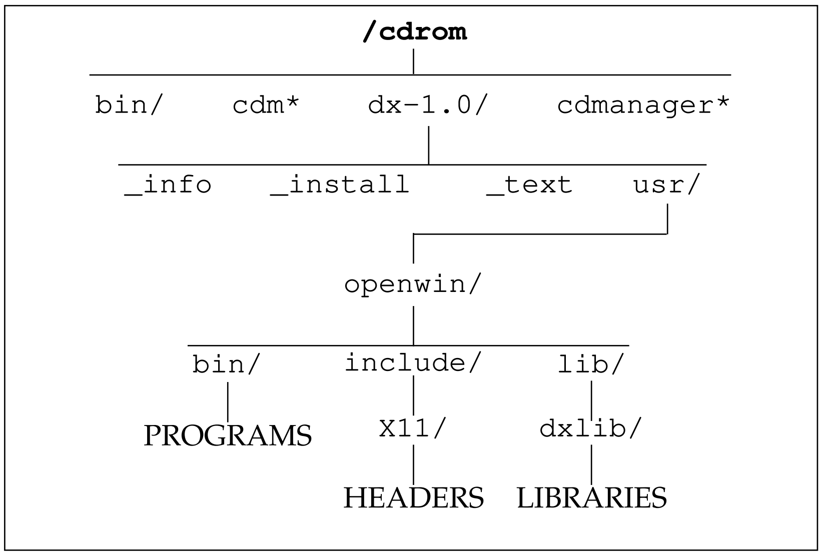

Figure 4-3 Direct Xlib Installation Directory Structure

Please refer to Chapter 2, “Installing Direct Xlib”, in the *Direct Xlib User’s Guide* (part number 800-7466-10) for detailed information.

Using Direct Xlib

Direct Xlib may be used in the following ways:

- With an individual application (i.e., using Direct Xlib on an application-by-application basis)
- With the entire desktop

To allow Direct Xlib to work properly under the widest range of conditions, it is beneficial to use the following procedures before starting the OpenWindows server:

1. Set the file descriptor limit in the shell from which you plan to start the server to the maximum value of 256 with the following command:

   ```
   % limit descriptors 256
   ```

   Direct Xlib currently requires one file descriptor for each direct X11 window. Increasing the limit before starting the OpenWindows server reduces the chance that applications will run out of file descriptors. If one logs out and logs back in, this step must be redone.

2. Find 10 Megabytes of disk space where the OpenWindows server has permission to read and write.

3. Start OpenWindows by giving it a path to this disk space.

   ```
   % openwin -sharedretainedpath directory_path_to_space &
   ```

   where ***directory_path_to_space*** is the directory path you have chosen. The path pointed to must not be a "`tmpfs`" mount.

To determine the system's current number of file descriptors, type:

```
% /etc/pstat -fT
183/582 files (files used/files supported)
273/312 inodes
51/138 processes
10776/40000 swap
%
```

Make sure that the quantity of files supported is not less than 400. If it is less than 400, you should reconfigure the kernel to support a larger number of file descriptors.

To run Direct Xlib on an application-by-application basis (assuming Direct Xlib is in the default installation location, and the application is dynamically linked to `libx11`), type:

```
% $OPENWINHOME/bin/dxlib myprog
```

where *myprog* is the name of the application. The program name may be followed with the usual command line arguments, if any. Direct Xlib activated in this manner is only active for the single application started with the `dxlib` script. The `dxlib` script must be executed with each application you want to use with Direct Xlib.

Running the following script with no application argument:

```
% $OPENWINHOME/bin/dxlib
```

will give you information about how to use Direct Xlib, its current status.

When using Direct Xlib with the entire desktop, one should be aware of the following limitations:

1. Do not use a `GENERIC_SMALL` kernel, because there are not enough file descriptors available for a large number of normal applications plus a large number of DGA applications.
2. NeWS-based applications may hang in a DGA environment (PageView is fine). This problem is limited to OpenWindows 3.0 and will be corrected in later versions.
3. Direct Xlib increases the total working set size by about 1.5 megabytes. On systems with limited memory, this may cause more paging than when Direct Xlib is not used.

To run Direct Xlib on the entire desktop, add the following to the `.xinitrc` file just before starting the window manager (`olwm` in the default case):

```
LD_LIBRARY_PATH=$OPENWINHOME/lib/dxlib/SunOS4.x:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

These lines should be added just before the line with the comment:

```
# OpenLook Window Manager
```

If you use a window manager other than `olwm`, place the line before the specific window manager is started.

Now if you type the following:

```
% printenv LD_LIBRARY_PATH
```

you will see the path `$OPENWINHOME/lib/dxlib/SunOS4.x` appended to the first part of the old `LD_LIBRARY_PATH`. Leave this in your value of `LD_LIBRARY_PATH` as long as you want Direct Xlib to be automatically used when you start programs from shells within OpenWindows or from the Desktop menu.

Even though Direct Xlib may be running on the entire desktop, it can be turned off on particular applications. To do so, set the `DIRECTX_DISABLE` environment variable to any non-zero value before starting the application:

```
% setenv DIRECTX_DISABLE 1
```

If Direct Xlib is installed somewhere else other than `$OPENWINHOME`, starting Direct Xlib gets to be a little different, but it can be easily done. Direct Xlib can also be used to support proprietary X11 extensions.

Please refer to Chapter 3, "Using Direct Xlib", in the *Direct Xlib User's Guide* for in-depth information and additional functionalities.

Includes, Compiles, and Links with Direct Xlib

When compiling Xlib X11 applications, you should continue to use the standard `include/X11` header files. The header files supplied with Direct Xlib are only needed by programmers who are developing proprietary X extension libraries or whose application code directly manipulates the X11 command buffer.

For dynamic links to `libX11`, you should continue to link to the standard libraries. Linking dynamically with Direct Xlib makes it difficult to move the application back and forth between standard Xlib and Direct Xlib environment. It causes the executable to reference libraries which may not be available through standard `libX11`. Any application which is dynamically linked with standard `libX11` can run with Direct Xlib. Links should include the C math libraries (`-lm`). When compiling and linking, remember to turn off Direct Xlib in your shell:

```
% setenv DIRECTX_DISABLE 1
```

For static links to `libX11`, you must link to the static version of Direct Xlib. Links should include the C math libraries (`-lm`).

If the application uses `libXext` and/or `libxinput` and is statically linked to either of them, the application must also link to the static Direct Xlib versions of `libX11`, and `libXext` and/or `libXinput` supplied with Direct Xlib.

Please refer to Chapter 3, “Using Direct Xlib”, in the *Direct Xlib User’s Guide* for more information.

Documentation

To use Direct Xlib features most effectively on your system, please refer to *Direct Xlib User’s Guide* (part number 800-7466-10). It covers the following topics in depth:

- Introduction (Chapter 1)

  Overview / System Requirements / Supported Primitives

- Installing Direct Xlib (Chapter 2)

  Installing Direct Xlib in different system configurations and environments, which include Standalone Workstations, Homogeneous Server Workstations, and Heterogeneous Server Workstations.

- Troubleshooting (Chapter 4)

  Error detection techniques for programmers and system administrators. Addresses problems that may arise during the installation of Direct Xlib or OpenWindows and in turning on DGA.

Languages 5

Dynamic Linking

Dynamic linking allows standard library functions and system calls to be linked into application programs at run-time. Unlike statically linked programs, a dynamically linked executable file does not contain code for all of the functions it calls. Instead, the dynamic linker resolves calls to standard (shared) library functions at run-time.

Because system calls can be implemented differently on different machines of the same architecture, this separation of application-defined versus shared library functions is crucial for conformance with the Application Binary Interface (ABI). Thus, dynamic linking is the preferred compilation method for System V Release 4 (SVR4).

The strategy for dynamic linking in SVR4 is similar to that in SunOS 4.*x*. This article provides an overview of dynamic linking concepts and describes the improvements made in dynamic linking between SunOS 4.1.*x* and SVR4.

When to Use Dynamic Linking

In general, dynamic linking should be used whenever possible. When a program is dynamically linked, the shared objects it uses are mapped into its virtual address space at run-time and paged into memory as needed. Different programs that run simultaneously can share the common object code.

When a program is statically linked, a copy of each object file is incorporated into the executable at compile time. Statically linked code cannot be shared, and may be duplicated any number of times in memory.

There are several reasons applications should be dynamically linked:

- Dynamically linked programs save disk space and system process memory because they share library code at run-time.
- Shared-library code can be enhanced without having to relink the applications that use it. The dynamic linker simply uses definitions from a new version of the shared library when resolving references to the shared code.
- As long as the pathname of a shared object is not hard-coded into an application, the shared object can be moved to a different directory without affecting that application.
- Dynamic linking increases portability; a dynamically linked application can run without recompilation on different machines of the same machine architecture under different implementations of SVR4.
- Dynamic linking contributes to overall system performance. When a number of applications share the same code, that code is not duplicated in memory. Only one page fault is needed to bring in the shared code, increasing the system performance.

When to Avoid Dynamic Linking

If an application calls only a few small library functions, the bookkeeping information used by the dynamic linker may take up more space in the executable than the code for those functions. Use the `size(1)` command to determine the difference.

Although a shared object's text is shared by all processes that use it, the associated data is not. Every process that uses a shared object typically receives a private copy of the shared object's entire data segment. If an application uses only a small portion of a shared library's text and data, executing the application may require more memory with a shared object than without one. A process that uses a very small portion of a shared library may save memory when statically linked to that library.

The Link Editor

Linking is the process of connecting symbols referenced in one module with definitions from other modules. The link editor searches your program and the libraries you link with for definitions of external symbols, and attempts to resolve those symbols.

The Static Linker

When creating a statically linked program, the link editor either completely resolves all external symbols by incorporating the associated program text, or fails with an error. When creating a dynamically linked program, the link editor allows external symbols to remain undefined; these remaining symbols are left to be resolved by a run-time invocation of the link editor, `ld.so`.

The static link editor searches for libraries in the directories you specify as follows:

- directories given in the `LD_LIBRARY_PATH` environment variable; this variable's value takes the form of a colon-separated list of pathnames
- each directory specified with a `-L` option to `cc(1)`, the C compiler
- next, if `LD_LIBRARY_PATH` contains a semicolon, those directories listed after the semicolon
- finally, the standard directories for object libraries (`/usr/lib`)

The Dynamic Linker

When a user runs a process that is dynamically linked, the dynamic linker maps the shared libraries used by the process into the process space at execution time before *main*() receives control. As calls are made to functions in the shared library, the dynamic linker intervenes to resolve references. This is called "lazy evaluation" and it happens only once for each function being called. When you set the environment variable `LD_BIND_NOW` to "1", all function references are resolved before *main*() gets control.

At run-time, the dynamic linker searches for libraries in the directories you specify as follows:

- directories specified in the LD_RUN_PATH environment variable. LD_RUN_PATH must be set before compiling the application. Changing LD_RUN_PATH at execution time won't make the dynamic linker search in the new places.
- directories specified in the LD_LIBRARY_PATH environment variable. Changing LD_LIBRARY_PATH at execution time *does* allow the dynamic linker to search in the new places.
- finally, the standard directories for object libraries (/usr/lib)

Shared Library Naming Conventions

A shared library's text segment is contained in a file with a suffix that starts with .so, followed by a major version number. (Unlike SunOS 4.1.*x*, there are no minor version numbers in SunOS 5.0.) For instance, libc.so.1 contains the sharable text of the C library, version 1. The version number distinguishes between incompatible versions of a shared library.

The version used at execution time must match that of the library that the application was compiled against. Otherwise, the run-time link editor's attempt to resolve the reference fails, terminating program execution.

*SunOS 4.1.*x

In SunOS 4.1.*x*, a shared library may have exported initialized data that is stored in a file with a name ending in a suffix beginning with .sa, and followed by the version numbers.

The .sa file is an ar-format (archive library – see ar(1)) that is statically linked to each application at compile time.

SunOS 5.0

There is no need for a .sa file in SunOS 5.0. The global initialized data is also stored in the .so file. Other naming conventions remain the same as in SunOS 4.1.*x*.

Building Shared Libraries

In both SunOS 4.1.*x* and Solaris 2.0, shared libraries are built using position independent code (PIC). This allows the code to be mapped to different locations within the virtual address space of different processes.

Position independent code cannot contain absolute virtual addresses. The global offset table holds absolute addresses in private data, thus making the addresses available without compromising the position independence and shareability of a program's text.

SunOS 4.1.x

To build a `.so` file in SunOS 4.1.*x*, use the following commands:

```
% cc -pic -c *.c
% ld -o libname.so.ma.mi -assert pure-text *.o
```

The compiler generates position independent code when the `-pic` option is set.

Make sure that `.c` files do not contain definitions of external global initialized data (they may contain references to such data, however). All such data definitions must be segregated into the `.sa` file. To build a `.sa` file:

1. Segregate the declaration of exported global initialized data from the sources for each object into separate source files.
2. Create a separate (static) archive library composed of these data objects and give it a name of the form:
   `lib`***name***`.sa.`***ma.mi***
3. Use `ranlib(1)` to place a symbol table in the `.sa` archive.

Solaris 2.0

Use the following commands to build a complete shared library in Solaris 2.0. There is no need to build a `.sa` file.

```
% cc -c -k PIC *.c
% cc -G -o libname.so.ma.mi *.o
```

When the -k PIC option is set, the compiler generates position independent code that requires the least possible amount of page modification at run-time. Position independent code can be loaded anywhere in a process's address space.

Object File Format

The new object file format under Solaris 2.0, Executable and Linking Format (ELF), replaces the old format and is a fundamental part of the ABI solution.

Under SunOS 4.1.*x*, Common Object File Format (COFF) is the output format generated by the assembler and link editor. The link editor makes `a.out` executable files. The `a.out` file format consists of: a header, the program text, program data, text and data relocation information, a symbol table, and a string table. When an `a.out` file is executed, three logical segments are set up: the text segment, the data segment, and a stack.

Table 5-1 shows the ELF file format under Solaris 2.0.

Table 5-1 Executable File Format under Solaris 2.0

File Offset	File	Virtual Address
`0`	ELF Header Program header table Other information	`0x00000`
`0x100`	Text segment . . . `0x2be00` bytes	`0x10100`
`0x2bf00`	Data segment . . . `0x4e00` bytes	`0x4bf00`
`0x30d00`	Other information . . .	`0x50cff`

The first text page contains the ELF header, the program header table, and the other information. The last text page holds a copy of the beginning of the data. The first data page has a copy of the end of the text. And the last data page may contain file information not relevant to the running process.

Dynamic Linking and the Binary Compatibility Package

The Solaris 2.0 Binary Compatibility Package (BCP) is an optional package that lets SunOS 4.1.*x* application run on Solaris 2.0 without recompilation or relinking if they meet the following restrictions. To run under the compatibility package, a SunOS 4.1.*x* application:

- must be dynamically linked
- must not access `libkvm` or `/dev/kmem`
- must not write directly to system files
- must not rely on customer-supplied drivers and/or `ioctls`
- must not trap directly into the kernel
- must use only publicized SunOS interfaces

If the BCP is installed on a machine, conforming SunOS 4.1.*x* and Solaris 2.0 applications can run simultaneously.

To use the Binary Compatibility Package, the user should set the `PATH` environment variable so that `/usr/ucb` appears first.

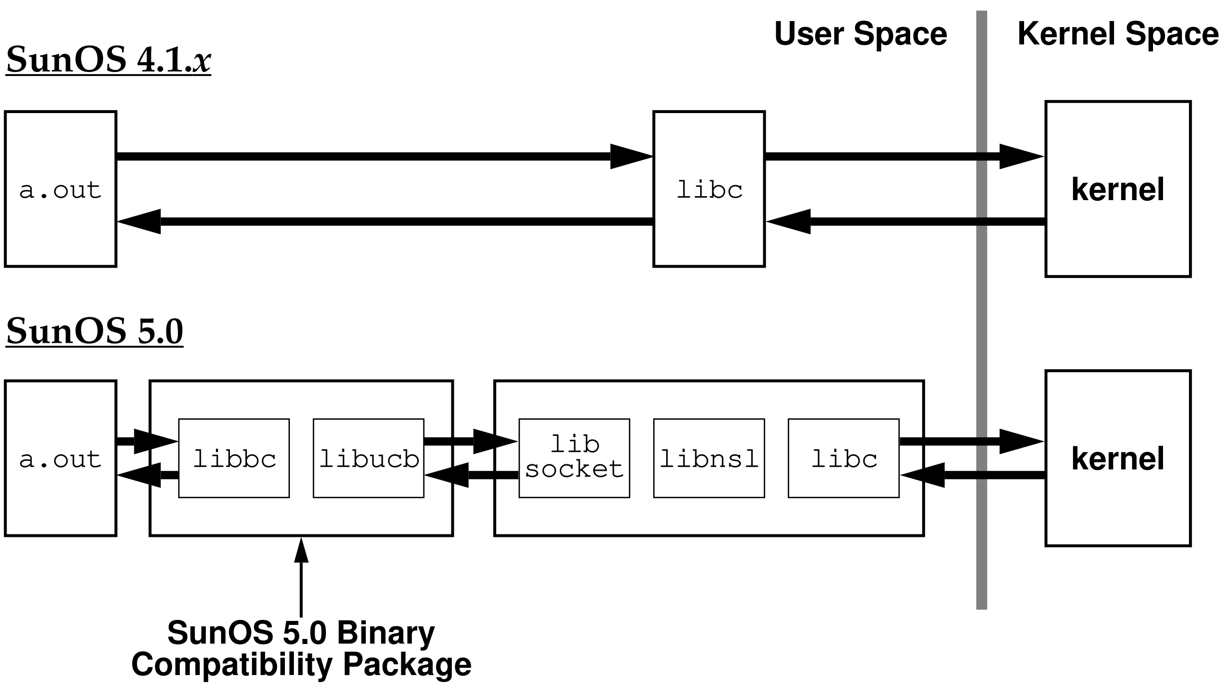

Figure 5-1 SunOS 5.0 Binary Compatibility Package

As shown in Figure 5-1, the Binary Compatibility Package includes `libbc.so` that is loaded at run-time by `ld.so`. All the calls are routed through `libbc`, which in turn calls appropriate routines in `libc`, `libsocket`, and `libnsl`.

An application must be dynamically linked with `libc` and `libnsl` on SunOS 4.1.*x* to take advantage of the Binary Compatibility Package. An optional Source Compatibility Package exists in Solaris 2.0 that can be used to recompile an application in Solaris 2.0 without any change to the source code. However, it is recommended that applications be ported to the native Solaris 2.0 environment.

For more information on the compatibility packages, read the *SunOS/BSD Source Compatibility Package Guide – SunOS 5.0* (part number 800-6443-10) and the *Binary Compatibility Package Guide – Solaris 2.0* (part number 800-6442-10).

Programmatic Interface to Dynamic Linking

SunOS provides a programmatic interface to the run-time linker which you can use to bind or unbind shared libraries during the course of program execution. This interface is useful when you are in the process of building a shared library for your application and the start-up time for your application is a significant performance factor.

An application should not have to be restarted every time a change is made to the shared library. Applications should wait for the latest build of the shared library to be complete and then use the programmatic interface to call the most recent implementation of the shared library.

The `dlopen()` routine allows you to get access to a shared library, which is bound to the process's address space. `dlsym()` returns the address of a given symbol within a shared library. `dlclose()` deletes the reference to a shared object. When the last reference is deleted, the shared object is removed from the process's address space. `dlerror()` can be used to obtain information about the last error occurring as the result of `dlopen()`, `dlsym()`, or `dlclose()`.

The sample program below illustrates the programmatic interface; the `pnum()` function is defined in the shared library `libshared.so.1`.

Programmatic Interface Sample Program

```
/* libshared.so */
#include <stdio.h>

. . .

void
pnum(num)
int num;
{
       printf ("num = %d\n", num);
}
```

The program code is:

```
#include <stdio.h>
#include <dlfcn.h>

main()
{
       void (*open_handle)(), (*access_handle)();
       int status;

       open_handle = dlopen ("./libshared.so", 1);
       if (!open_handle) {
              printf ("failed to set shared library in the address space\n");
              exit (-1);
       }

       access_handle = dlsyn (open_handle, "pnum");
       if (!access_handle) {
              printf ("failed to get address of function pnum\n");
              exit (-1);
       }

       access_handle (10);
       status = dlclose (open_handle);
       if (status) {
              printf ("dlclose failed\n");
              exit (-1);
       }
}
```

Increasing Shared Library Performance

You can enhance shared library performance by doing the following:

- Minimize the library's data segment by using automatic or static variables. This avoids allocating permanent storage, since space for local variables is allocated on the process stack with every invocation of a subroutine. Space for every global variable in a library is allocated in each process, whether the process uses those variables or not.
- Use functional interfaces rather than global variables whenever possible. Maintaining library interfaces and code is easier when using a functional interface, and may eliminate your need for global variables entirely.
- Make every effort to minimize paging. One method is to exclude infrequently used routines on which the library itself does not depend. It is a good idea to arrange the shared library's object files so that frequently used functions do not unnecessarily cross page boundaries. Arranging the typical process to avoid cache entry conflicts helps to increase shared library performance as well by avoiding hardware thrashing.

Other Considerations

Dynamic linking does not report errors when multiply-defined symbols exist. In other words, an application that contains multiple definitions of a function or a same-sized data object that reside within a different shared object or within a dynamically linked executable and different shared objects will not be flagged by the dynamic linker.

When linking a set-user of set-group ID program, the dynamic linker ignores any directories specified by `LD_LIBRARY_PATH` that are not "trusted." Trusted libraries are built into the dynamic linker and cannot be modified by the application. Currently, the only trusted directory is `/usr/lib`.

Other Reading

"Dynamic Shared Libraries," Jeffrey Peacock, *Unix Review*, vol. #9 No. 5.

Programmer's Overview, part number 800-3847-10, Sun Microsystems, Mountain View, CA.

System V Application Binary Interface: SPARC Processor Supplement, Unix Software Operation, Prentice-Hall, Englewood Cliffs, NJ.

System V ANSI C and Programming Support Tools, Prentice-Hall, Englewood Cliffs, NJ.

Tools 6

SunSolve

SunSolve online service and support solutions give you answers to the most common problems facing Sun system users. And you get those answers where they're most useful—right at your workstation. With SunSolve, you get the latest and most important system information to help keep your workstation up and running—common symptoms and resolutions, software bug reports, bug and patch descriptions, optional integration with the AnswerBook document database, and other useful databases. Plus you get the control that comes with immediate, round-the-clock access to these sources.

SunSolve comes to you on sturdy, reliable CD-ROM. You get the entire package on one disc instead of many tapes or stacks of paper—and there's no need to sacrifice valuable hard-disk space, because SunSolve has flexible installation options. What's more, SunSolve software on CD-ROM is as easy to install as it is to use.

Software support contract customers will be provided with both the initial SunSolve distribution and an updated SunSolve CD-ROM every quarter at no additional charge. Because SunSolve comes to you regularly, you have access to the most up-to-date, proven solutions from the industry's top engineers. Your information is always current.

If you need more frequently updated information, you can use *SunSolve Online*, which is accessible by modem. This service replaces the previous online system called Online Support System (OSS). Password and login information for SunSolve Online is exactly the same as OSS password and login information.

The software makes documentation that once filled entire bookshelves available to you online. You can access current and back issues of the *Customer Distributed Bugs List* and *Software Technical Bulletin*, plus various manuals, guides, and technical support articles—all onscreen, exactly as they appear in hard copy.

Because it's based on the OPEN LOOK graphical user interface, SunSolve software is intuitive and easy to use. Its two powerful tools, SearchTool and DocViewer, make navigating through SunSolve simple. And the software includes a variety of pull-down menus, help buttons, and an online help facility to guide you.

SearchTool

With SearchTool, you can examine databases by typing in either a single key word or string. Type "`file system full`", for instance, and SearchTool will find entries pertaining to a full file system.

SearchTool also performs complicated Boolean searches, a powerful capability that helps you narrow the search field to just the information you need. For example, type in "`file system full`" along with "`root`", and SearchTool brings up information about a full file system and the root partition—only the information you ask for, with no extraneous data.

It's also possible to save instructions you use frequently into files. Instead of having to remember and type in information from prior searches, you simply call up the file and launch it with a click of the mouse.

DocViewer

DocViewer, SunSolve software's onscreen document viewing tool, lets you call up page-for-page replicas of printed documentation. Start with the tool's "Describe" button for a listing and synopsis of all the documents at your fingertips. Then simply bring the document up by clicking the mouse on whichever document you choose. Page through the documents forward or backward, or specify particular pages that interest you. It's that easy.

DocViewer also enables you to print documents in an array of formats, including U.S. letter, legal, A3, A4, A5, and B5. You can print all or any part of your document by clicking the mouse, or if you prefer, print to a file.

Features	Benefits
• Online technical solutions developed by Sun experts	• Let you access an extensive database collection, including symptoms and resolutions, bug and patch descriptions, *Software Technical Bulletin*, and more • Optional integration with AnswerBook documentation

SunSolve offers comprehensive toolsets to access solutions, including:

Features	Benefits
• SearchTool	• Searches for single key word or string • Can search through any or all available databases • Performs Boolean searches • Lets you save often-used instructions to a file
• DocViewer	• Gives you exact replicas of hard copy documentation • Prints documents in a variety of formats • Prints to files
• Easy-to-use OPEN LOOK interface	• Employs a simple, familiar graphical user interface • Provides buttons and pull-down menus • Makes online help available
• Delivered on CD-ROM	• Easy to install • Sturdy and reliable • Flexible installation options let you save hard disk space
• Updated regularly	• Ensures that you have the most current support information

Software Release Versions 7

Current Sun Software Products and Release Versions

The following tables contain lists of Sun software products and their respective release versions, current as of June 30, 1992. These tables appear monthly in the STB for your convenience.

Please contact your sales representative if you would like to update the release level of a Sun software product you already use, or wish to purchase another product. Use the tables to determine whether your release is the most current.

Please note the following software product change(s):

a. As of the release of Solaris 2.0, the C compiler will be unbundled.

Operating Systems

Product Name	Current Release
SunOS (sun-4, sun-4c)	4.1.2
SunOS (sun-3, sun-3x)	4.1.1_U1

Window Systems & Desktop Utilities

Product Name	Current Release
OpenWindows	3.0
DeskSet Environment (SunView)	1.1

Sun Server & Database Software

Product Name	Current Release
Sun Database Excelerator (SunDBE)	1.2
SimplifySQL	1.1
Sun SHIELD ARM and ASET	1.0
Backup Copilot	1.0
Online: DiskSuite	1.0
Online: IPI Dual Port Tools	1.0

Graphics Software

Product Name	Current Release
SunVision	1.2
XGL	2.0
SunGKS	4.1
SunPHIGS	2.0
SunPEX	1.0
Direct Xlib	1.0

Network Management

Product Name	Current Release
SunNet Manager	2.0
SunNet License	1.1

Software product availability is current for U.S. releases.
Availability may be delayed outside the U.S.

Software Development

Product Name	Current Release
Sun Ada Development Environment	1.1
Sun C	1.1
Sun C++	2.1
Sun COBOL	1.0
Sun COBOL Runtime System	1.0
Sun COBOL Forms-2	1.0
NetISAM	1.0
OpenWindows Developer's Guide (DevGuide)	3.0
ONC / RPC Application Toolkit	1.0
Sun FORTRAN	1.4
Sun Pascal	2.1

Other Products

Product Name	Current Release
NeWSprint	2.0
VideoPix	1.0
PC-NFS	4.0
PC-NFS LifeLine	2.0
PC-NFS Advanced Telnet	1.0
PC-NFS Programmer's Toolkit	2.0
SunPC	3.0

Software product availability is current for U.S. releases.
Availability may be delayed outside the U.S.

Connectivity Products

Product Name	Current Release
SunLink CG3270	7.1
SunLink SNA3270	7.0.1
SunLink BSC3270	6.1
SunLink BSCRJE	6.0
SunLink SNA Peer-to-Peer	7.0.1
SunLink Local 3270	6.1.1
SunLink Channel Gateway	7.1
SunLink DNI	7.0
SunLink TE320	7.0
SunLink OSI	7.0.1
SunNet MHS	7.0
SunLink IR	6.0
SunLink X.25	7.0
SunLink TRI/S	2.0
SunLink HSI/S	1.1
SunLink FDDI/S	1.0
Serial Parallel Controller	1.1
SunLink MCP	6.0
SunLink FDDI/DX	2.0
DOS Windows	1.0.1

Software product availability is current for U.S. releases.
Availability may be delayed outside the U.S.

Updated Sun Operating System Support Matrix

Sun's policy is to provide hardware and software support for out-of-production or transitioned systems and the latest compatible software for those systems for a minimum period of five years from the last order date.

The details of Sun's software support policy have been published in a Sun Operating System Support matrix in past STB issues. As illustrated in the support matrix, older versions of the operating system generally receive lower levels of software support than newer versions. Thus, we recommend that you upgrade your system to the most current versions to ensure you receive the best quality of support from Sun.

Since we continually enhance the operating system to meet our customers' needs, we promised to keep you abreast of updates to the support matrix via the STB. Now that SunOS 4.1.2 is available, we provide an updated matrix.

Sun uses "support levels" to describe its software support policy. Please note that the services described in these levels are subject to local availability—contact your local service manager should you have any questions.

Level 1 Support

Level 1 offers the complete array of services contained in Sun's Personal AnswerLine Service, AnswerLine Service, and Software Update Service (see Note ①). Level 1 support also includes the full availability of options such as SunStart™, Sun's educational services, and supplementary consulting products and services.

Level 2 Support

Level 2 includes Sun's Personal AnswerLine and AnswerLine levels of phone support, limited availability of software and documentation, bug report submission and tracking, and emergency fixes and workarounds (see Note ①). Level 2 support also includes availability of supplementary consulting products and services.

Level 3 Support

Level 3 provides Time and Materials (T&M) support only.

Note ① — Software updates will only be available for the most current release. When a version of software has been designated as the final release for a platform, only critical bug fix releases will be available for that platform.

Sun Operating System Software Support Matrix

The following tables illustrate Sun's OS software support matrix. See Note ② below regarding this section.

For Sun-2	3.5	4.0/4.0.1	4.0.3	(see Note ③)
1/91-1/92	level 3	level 3	level 2	
1/92-1/94	level 3	level 3	level 3	

For Sun-3	3.5	4.0/4.0.1	4.0.3/4.0.3e	4.1	4.1.1	(see Note ③)
1/91-1/92	level 3	level 3	level 2	level 1	level 1	
1/92-1/95	level 3	level 3	level 3	level 2	level 2	
1/95-1/96	level 3	level 3	level 3	level 3	level 3	

For Sun386i	4.0.1i	4.0.2i	(see Note ③)
1/91-1/92	level 2	level 1	
1/92-1/95	level 3	level 2	
1/95-1/96	level 3	level 3	

For SPARC	3.5	4.0/4.0.1	4.0.3/4.0.3e	4.1	4.1.1	4.1.2
1/91-present	level 3	level 3	level 3	level 2	level 1	level 1

Note ② — The above levels 1 and 2 are for contract customers only. Non-contract customers have the option to purchase support on a Time and Materials basis or use Sun's consulting services. The then-current unbundled products will be supported at the same level as the customer's operating system. Support for emergency fixes and workarounds may require the customer to upgrade to a more current version of the product.

Note ③ — These will be the final releases for these product lines, with the exception of critical bug fix releases. Sun's future SVR4-based operating system will function primarily on SPARC processor-based systems (Intel-based support for SVR4 is available with SunSoft's INTERACTIVE UNIX product line). Interoperability between Sun-3 or Sun386i systems and SPARC SVR4 systems will be fully supported through services such as NFS, NIS, remote login, and the like.

In addition, Sun is committed to supporting Sun-3 diskless clients running SunOS 4.1.1 on Solaris 2.0 servers.

Notes & Comments 8

World Hotlines

Sun Customers throughout the world have service hotlines available for both software and hardware support questions. The service hotlines are shown below. If your country is not shown in the table, please phone your local Sun sales office. World hotlines are divided into those for the U.S.A., Europe, and Intercon.

United States

All	**1-800-USA-4SUN** (**1-800-872-4786**)

Europe

Finland	Sun Microsystems Oy	**+358-0-502-2700**
France	Sun Microsystems France SA	**+33-1-30-67-50-80**
Germany	Sun Microsystems GmbH	**+49-89-46008-321**
Italy	Sun Microsystems Italia SpA	**+39-6055228**
Netherlands	Sun Microsystems Nederland BV	**+33-501234**
Spain	Sun Microsystems Iberica SA	**+34-1-5560243**
Sweden	Sun Microsystems AB	**+46-8-623-9100**
Switzerland	Sun Microsystems (Schweiz) AG	**+41-1-825-7272**
	Sun Microsystems (Suisse)	**+41-22-64-4735**
United Kingdom		
	Sun Microsystems UK Ltd	**+44-276-691974**

Intercon

***Canada*:**

	Quebec District Montreal, Quebec	**(514) 744-9240**, **(800) 361-1554**
	Ottawa District Ottawa, Eastern Canada	**(613) 723-8112**, **(800) 465-4415**
	Central District Toronto, Central Ontario	**(416) 490-9490**, **(800) 263-1680**
	Prairies District Manitoba, Saskatchewan, Alberta, Calgary, Western Canada	**(403) 262-6722**, **(800) 661-9256**
	Edmonton District Edmonton	**(403) 421-7264**, **(800) 661-3643**
	B. C. District Vancouver, British Columbia	**(604) 684-4120**, **(800) 663-0440**
	National CSS/Education	**(416) 477-6745**

Australia	Sun Australia	**(008) 02-4417**
New Zealand	Auckland Wellington	**64-9-525-2749** **64-4-499-2344**
Hong Kong	Sun Asia	**852-802-4188**
Japan	Nihon Sun	**044-819-3121**

Latin America: Contact the Latin American hotline at ICON headquarters:
U.S. Office **(415) 688-9160**

Within ***Brazil***, you may also contact:
Sun do Brasil **55-11-887-9011**

Caribbean: Call your local distributor.

All Other Countries
Call **(415) 688-9372**, which is a U.S. number.

Reporting Bugs

This article contains three sections for submitting bugs. The first section describes procedures to use within the United States and Canada. The second section describes procedures within Europe. The third section describes procedures within ICON.

Submitting Software Bugs: United States and Canada

This section contains information on reporting bugs within the U.S., for customers holding and not holding support contracts.

Sun's United States Solution Center accepts software bug reports from Sun users via electronic mail and by phone. The method you use to submit a bug report varies with your needs.

U.S. users holding support contracts can report bugs to US Solution Center via the **(800) USA-4-SUN** phone hotline. Canadian users holding support contracts should call their local support center. The U.S. Solution Center phone hotline is the fastest way for a customer to find out if a problem is known and if a workaround exists. The status of previously-reported bugs can also be obtained in this way.

U.S. customers holding support contracts can also submit bug reports electronically to the address *sun!hotline* (or *hotline@Sun.COM*). Canadian customers need to use *hotline@Canada.Sun.COM*. This method generates a service order, and can be used when lines of code or other information difficult to relay over the phone is needed to describe the bug.

Customers who do not hold Sun software support contracts can report bugs via electronic mail to the address *sun!sunbugs* (or *sunbugs@Sun.COM*). These reports are reviewed periodically to determine proper disposition. Those reports determined to be from supported customers are forwarded to the U.S. Solution Center for handling. Reports from customers who cannot be verified as holding a support contract are reviewed by Sun's engineering and support personnel. An internal bug report is generated if the reported bug is new and verifiable.

Finally, customers not holding software support contracts may call the **(800) USA-4-SUN** phone hotline to report a problem and request support on a Time and Materials (T&M) basis. Canadian customers should call their local support center. In this case, please have a Purchase Order (PO) number for billing purposes.

Summary: United States and Canada

For U.S. contract customers, **(800) USA-4-SUN** is the best method to report bugs. Canadian contract customers should report bugs to their local support center. The hotlines to use appear monthly in the note entitled "World Hotlines."

The electronic mail address *sun!hotline* (or *hotline@Sun.COM*) is available for U.S. customers to submit materials that are difficult to relay over the phone. Canadian customers should use *hotline@Canada.Sun.COM* .

For non-contract customers, the electronic mail address *sun!sunbugs* (or *sunbugs@Sun.COM*) is available to report bugs. To help us serve you better, please include the following information with all electronic mail reports:

- Your name
- The name and address of your organization
- Your Sun site code, if available
- Your workstation model and serial number
- The software release(s) you are running
- A description of the problem that you are experiencing

Submitting Software Bugs: Europe

This section contains information on reporting bugs within Europe, for customers holding and not holding support contracts.

Procedures for submitting bugs are similar to those used in the United States. All customers should use their local country Answer Center to report bugs, with contract customers receiving a specific follow-up.

Sun customers not holding software service contracts can call their local Answer Center, and will need to provide a Purchase Order (PO) number at the time of the call.

Summary: Europe

To help European service centers serve you better, please include the following information with all electronic mail reports:

- Your name
- The name and address of your organization
- Your Sun site code, if available
- Your workstation model and serial number
- The software release(s) you are running
- A description of the problem that you are experiencing

Finland

Report bugs to the Finland Answer Center at the postal address shown below.

Hotline
Sun Microsystems Oy
Sinikalliontie 12
SF-02630 Espoo
Finland

Hotline Telephone: +358-0-502-2700
Telefax: +358-0-502-2800

E-mail Address: *Hotline@Finland.Sun.COM*

Special Dispatch Arrangements:
Provide Dispatch with the Contract number.

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

France

Report bugs to the France Answer Center at the postal address shown below.

Service Hot Line
Sun Microsystems France SA
13, Avenue Morane Saulnier
BP 62
78142 Velizy
France

Hotline Telephone: +33-1-30-67-50-80
Telefax: +33-1-30-67-53-70
Minitel: 36 14 code SUNFRANCE

Special Dispatch Arrangements:
Please provide Dispatch with System serial number *or* Contract number.

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis, and order this support at the above address.

Germany

Report bugs to the Germany Answer Center at the postal address shown below.

Hotline
Sun Microsystems
Bretonischer Ring 3
8011 Grasbrunn
Germany

Hotline Telephone: +49-89-46008-321
Telex: 5218197 sun
Telefax: +49-89-46008-320

E-mail Address: {*sunuk,unido*}*!sunde!hotline* or *hotline@Germany.Sun.COM*

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

Italy

Report bugs to the Italy Answer Center at the postal address shown below.

Hotline
Sun Microsystems Italia SpA
Centro Direzionale Colleoni
Palazzo Andromeda (Ingresso 1)
Via Paracelso 16
20041 Agrate Brianza
Milan
Italy

Telephone (Milan Answer Center): +039-6055228
Telex: 326423 MBOX SUNITA
Telefax: +039-6056764

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

The Netherlands

Report bugs to The Netherlands Answer Center at the postal address shown below.

Sun Microsystems Nederland BV
Computerweg 1
3821 AA Amersfoort
Netherlands

Hotline Telephone: +33-501234
Telefax: +33-553058
Telex: 79297 sun nl

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

Spain

Report bugs to Spain Answer Center at the postal address shown below.

Sun Microsystems Iberica SA
Plaza Pablo Ruiz Picasso s/n
Torre Picasso Planta 25
28020 Madrid
Spain

Hotline Telephone: +34-1-5560243
Telefax: +34-1-5564097

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

Sweden

Report bugs to the Sweden Answer Center at the postal address shown below.

Sun Microsystems AB
Box 51
S-164 94
Kista
Sweden

Hotline Telephone: +46-8-623-9100

E-mail Address: *hotline@Sun.se* or *hotline@Sweden.Sun.COM*

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

Switzerland

Report bugs to the Switzerland Answer Center at one of the postal addresses shown below.

(*German*)
Sun Microsystems (Schweiz) AG
SAPostfach
Eschenstrasse 8
CH-8603 Schwerzenbach
Switzerland

Telephone: +41-1-825-7272
Telefax: +41-1-825-7222

E-mail: *hotline@Swiss.Sun.COM*

(*French*)
Sun Microsystems (Suisse)
Route des Avouillons 12
CH-1196 Gland
Switzerland

Telephone: +41-22-64-4735
Telefax: +41-22-64-1176

E-mail: *answerline@Swiss.Sun.COM*

Special Dispatch Arrangements:
Provide Dispatch with the Contract number and System serial number.

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

United Kingdom

Report bugs to the UK Answer Center at the postal address shown below.

Sun Microsystems UK Ltd
Watchmoor Park
Riverside Way
Camberley
Surrey
GU15 3YL
England

Hotline Telephone: 0276-691974
Telefax: 0276-681546

Special Dispatch Arrangements:
Please provide Dispatch with either the System serial number *or* Contract number.

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

Submitting Software Bugs: Intercon

This section contains information on reporting bugs within Intercon, for customers holding and not holding support contracts.

Procedures for submitting bugs are similar to those used in the United States. All customers should use their local country Answer Center to report bugs, with contract customers receiving a specific follow-up.

Sun customers not holding software service contracts can call their local Answer Center, and will need to provide a Purchase Order (PO) number at the time of the call.

Summary: Intercon

To help Intercon service centers serve you better, please include the following information with all electronic mail reports:

- Your name
- The name and address of your organization
- Your Sun site code, if available
- Your workstation model and serial number
- The software release(s) you are running
- A description of the problem that you are experiencing

Detailed information for individual countries follows.

Australia

Report bugs to the Australian Answer Center at the postal address shown below.

Hotline
Sun Microsystems Australia Pty. Ltd.
PO Box 320
Artarmon, New South Wales 2064
Australia

Hotline Telephone: (008) 02-4417
Telefax: 02 436 1084

E-mail Address: *hotline@Aus.Sun.COM*

Special Dispatch Arrangements:
Please provide Dispatch with either the System serial number *or* Contract number

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

Japan

Report bugs to the Nihon Sun Answer Center at the postal address shown below.

Nihon Sun Microsystems K.K
KSP Technical Center - Answer Center
KSP R&D Bldg. B-1F-112
Sakato 100-1, Takatsu-ku,
Kawasaki, Kanagawa, 213
Japan

Hotline Telephone: 044-819-3121, 044-819-3111
FAX: 044-819-3120

Special Dispatch Arrangements:
Provide Dispatch with the Contract number and Serial Number

Arrangements for Non-Contract Customers:
Please provide a valid PO number for billing on a Time and Materials (T&M) basis.

New Zealand

Report bugs to the New Zealand Answer Center at the number shown below.

Hotline Telephone: 4-499-2344

Your STB Subscription

Your company's software service contract includes a monthly issue of the STB. Each month, the copy of your STB is mailed to your company's primary contact person or department named in the contract. Sites with more than one contract may receive more than one STB copy, depending on how the contracts are arranged.

Duplicating Your STB

Your primary contact person or department may duplicate this "master" STB copy for all Sun workstation end-users. As long as you duplicate copies and route them internally only, there are no copyright infringement problems.

This limited permission for duplication is for your convenience only, however, and does not include any duplication for resale, for distribution outside your company, or for distribution to employees of companies not having a Sun software service contract.

Multiple STB Subscriptions

You can now order extra copies of the STB for a low monthly fee for each additional copy. These extra copies will sent to the same address as your "master" STB copy. Please contact the e-mail addresses shown below, or contact your sales representative for additional monthly STB copies.

How to Order Additional STB Subscriptions

If you have any questions regarding your STB subscription or would like to order an additional subscription, please contact your local Sales/Service Office or Sun Authorized Distributor.

Further STB Questions

If you have any questions or comments regarding the STB, including corrections, please send your comments to *stb-editor@Sun.COM* or *sun!stb-editor*.

Index

O

P

S

X