Operating System Threads

by Joe Eykholt and Dan Stein

The SunSoft Developer Engineering team is dedicated to helping developers migrate their software applications to SunSoft products, and will provide the answers to your questions. Please submit queries to us via E-Mail at dev_eng@Sun.COM. We will attempt to answer all questions posted to us, but we may not have room to print all responses.

Is Solaris 2.2 I/O subsystem fully multithreaded? (i.e., is the I/O symmetric or bound to one CPU)?

The I/O subsystem is fully multithreaded and symmetric on all existing platforms. Any CPU may initiate I/O. However, the method of delivering I/O interrupts varies. On some platforms some devices will only interrupt a certain CPU. Neither drivers or streams modules are bound to any particular CPU.

What support is provided for third party vendors who write STREAMS modules?

The Solaris *Writing Device Drivers* document and the DDI manpages tell how to properly lock and synchronize drivers and STREAMS modules. The interface to create separate threads in a driver is being documented and will be released with the next major release of the DDI.

Are single-threaded drivers supported in Solaris 2.2? Will existing drivers work? What is the implementation, i.e., how is single-threaded driver support provided?

There is some support for what we call 'MT-unsafe' drivers. Whenever such a driver is active, it holds a global mutex, thus preventing any other thread from entering that driver or any other MT-unsafe driver. There are so many restrictions on MT-unsafe drivers that we've decided not to document this support. Driver writers must provide their own locking.

What synchronization primitives are used within the Solaris 2.2 kernel (i.e., advisory processor locks, spin locks, sleep locks)?

Mutexes and condition variables, primarily, as well as multiple-readers/single-writer (readers/writer) locks, and semaphores. Mutexes are mostly of an adaptive type which spin if the owner is running and suspend if the owner is suspended or preempted. We also have spin-type mutexes for use by high-level interrupts which are not blocked by the dispatcher.

Is the Solaris 2.2 kernel fully symmetrical? Are any specific system calls, instructions, etc. required to execute on a particular processor?

It is fully symmetric. Some hardware platforms may direct certain interrupts to a specific processor, but nothing in the OS requires this.

What type of locking strategy was used in designing the Solaris 2.2 kernel (for example, was a hierarchical scheme used)?

Locks protect data (not modules or functional blocks). Where two locks must be held at the same time, a hierarchy is established by convention. Rather than impose runtime checking, we built a tool that analyzes source files for lock hierarchy violations. The idea is to catch the lock hierarchy violations before they happen.

How are interrupts handled when the interrupt handler owns a lock and a new interrupt is received?

Most interrupts are handled by their own thread. If a new interrupt occurs which needs a lock already held by an interrupt thread, the new interrupt suspends or spins on the lock until the first interrupt releases the lock. Priority inheritance is used to expedite the completion of the code holding the lock.

How does the interrupt handler know that it already owns the lock?

The handler cannot already hold the lock, but another interrupt thread may. The owner of the lock is tracked by the synchronization object (mutex) code, but not communicated to the interrupt handler.

If a thread attempts to lock a mutex it already holds, the system panics. This is always the result of a coding error (it violates our specification).

Are any lock statistics collected from sample runs of Solaris 2.2; and if so, can you elaborate on their statistical analysis?

Such data exists, but not in a ordered fashion for us to be able to share with you.

Is Solaris 2.2 configured to execute on both uniprocessor and multiprocessors without recompilation (i.e., does a single binary version run on both types of machines)?

We support five varieties of SPARC machines:

> Sun4 (the 4/260, 4/110, 4/390, 4/490 series, etc.)
> Sun4e (the SPARCengine 1/e VME 6u boards)
> Sun4c (SPARCstation 1, 1+, 2, IPC, IPX, ELC, etc.)
> Sun4m (Classic, LX, SPARCstation 10, SPARCserver 690, etc.)
> Sun4d (SPARCcenter 2000, etc.)

Only the Sun4m and Sun4d are multiprocessor architectures. All Sun4m and Sun4d machines, UP and MP, run the same binary.

What requirements were set forth for uniprocessor versus multiprocessor executions of Solaris 2.2?

We required uniprocessor performance to be satisfactory when measured against our previous UP-only releases. However, Solaris 2.2 has many different features which makes it hard to compare to Solaris 1.

Are any atomic system calls provided, i.e., calls that guarantee only one instance of execution at a time? If so, what are they?

With Solaris 2.2 we provide a new set of thread synchronization primitives which permit application level threads to coordinate their activity. These primitives do not result in a system call unless there is contention. We also support system V IPC which is a system call interface and is not nearly as efficient as the new thread synchronization primitives.

What changes have been made to the PROC file system to support multiprocessing?

There are hooks to control individual LWPs in another process. Information has been added to the prstatus_t structure to determine which CPU a LWP last used and its processor binding.

What changes were made to system calls to support multiprocessing (i.e., what functions had to be changed)?

No standard system call interfaces were changed.

What changes were made to user-level libraries to support multiprocessing (i.e., what functions had to change)?

Nothing for multiprocessing, per se. We did extensive work to allow multithreading (applicable even to uniprocessors), but as far as we know, no standard ABI interfaces needed to change because of this.

What are the scalability requirements of Solaris 2.2?

Solaris scales from laptops to large servers today. These requirements are set by customers, depending on their environment and applications.

How does the paging policy of the virtual memory subsystem support multiple light-weight threads?

With threads, a process can generate multiple page faults in parallel. We improved our page faulting code to support this. The paging policy in Solaris 2.2 is otherwise unchanged.

What file system drivers have been multithreaded?

The code that implements our filesystems in the kernel is fully multithreaded.

What extensions are provided in Solaris 2.2 for priority control commands to support multiprocessing? For example, does priocntl(1) affect threads in addition to processes?

To answer this question requires some knowledge about the model of multi-threading in Solaris 2.2. Threads in Solaris 2.2 are user-level entities that are implemented by a user-level library. It is this user-level library that schedules threads onto kernel level entities, called LWPs, that really represent execution resources. The priocntl() interface changes the scheduling behavior of LWPs.

Applications use the thread interfaces. Most threads in an application do not require special scheduling behavior. They can execute on a symmetric pool of LWPs. If the application needs to create a special thread that has unique scheduling behavior like real-time, then the application can create a thread that is permanently bound to an LWP. A priocntl() with the appropriate parameters can be done from this bound thread to change the bound thread's scheduling behavior to real-time.

In addition, priocntl() can affect the entire process. This means all the LWPs in the process will be changed to have the selected scheduling behavior.

Describe the extensions to the ps(1) command to support multiprocessing.

None.

What user level commands in Solaris 2.2 are fully multithreaded?

None.

What extensions were made to system activity reporting commands (sar, sa1, sa2, sadc, sadp, times) to support multiprocessing?

None.

What administrative commands were added to Solaris 2.2 for supporting the configuration management of multiple processors?

We added psrinfo(1M) which gives the status of the processors present and psradm(1M) which allows the superuser to enable and disable processors. We also added pbind(1M) which allows users to bind processes (but not individual LWPs) to CPUs.

What modifications/extensions were made to the TLB functionality to enable multiprocessing?

Not applicable. The TLB hardware support varies from platform to platform.

What is the size of the proc structure?

Approximately 1520 bytes (measured on a more recent version).

What is the size of the user structure?

This is included in the proc structure.

Can the user structure be swapped?

The Solaris model of multithreading has changed the process model. A process can have multiple threads of control. The user and proc structure in the traditional UNIX process model together represented a single thread of control. To implement multiple threads of control within a process required that a thread of control be extracted from the proc and user structure. This new structure is called an LWP which represents a execution resource. LWPs should not be confused with threads which are user-level entities.

When a thread of control is extracted from the process model, there is no reason to make a distinction between a proc and user struct. Both the proc and user structure represent the state shared by threads within a process. The interesting user state is now in the LWP struct which is swappable.

The LWP structure is 8k bytes where most of the bytes are reserved for its kernel stack.

Is processor affinity supported for light-weight processes?

Yes. Loose-affinity is automatic, and explicit binding can be set with the processor_bind(2) system call or pbind(1M).

What are the scheduling classes?

Timesharing (TS) and real-time (RT) and system class (not exposed).

Does a light-weight process executing in kernel mode determine which thread to run, or does the user library decide?

LWPs represent execution resources which are scheduled by the kernel onto CPUs. Threads are also like execution resources; however, they are scheduled by a user library onto LWPs. This is usually referred to as a two-level model of multithreading. These threads have full access to system services. They are truly independent threads of control within a process. The process does not block when one of its threads block.

Are spin locks provided at the user level?

User level spin locks have a serious disadvantage and are not considered a good programing practice. The thread that owns a spin lock at any time can be time-sliced or preempted. When this happens the application can burn CPU cycles spinning until the owner of the spin lock is resumed. We decided not to support them. They are very simple to implement when they are needed.

How many subsystems are available in the kernel? Are any implemented with monitors, or are all fully symmetrical?

All are fully symmetric and some are implemented with monitors.

What are the differences between Sun's thread implementation and USL's?

First, UI adopted the Solaris model of multithreading. From the API point of view we are identical. USL's implementation of user-level threads is based on an early version of Sun's implementation.

User Level Threads

How does an application programmer use the threads library?

Threads are integrated into Solaris 2.2. An application using threads simply links with the threads library by adding “-lthread” to the compile line.

What are the names of the thread primitives in Solaris 2.2 (for example, thread_create())?

Refer to the Solaris 2.2 manpages.

What thread libraries are currently available in Solaris 2.2 and what libraries are planned for the future?

We have one threads library called libthread.so.1 in Solaris 2.2. It represents the UI thread interfaces. When POSIX 1003.4a becomes a standard, we will have a pthreads library. Both the pthreads library and libthread.so.1 will interoperate.

Does the Solaris 2.2 user thread package conform to POSIX 1003.4a? If so, Draft 6 or 7?

No. There have been major changes between drafts which makes having a implementation that conforms to either of these drafts very uninteresting.

What new process states are provided for in the Solaris 2.2 thread model?

None.

What are the thread scheduling primitives?

Refer to the Solaris 2.2 manpages.

What priorities are provided for in the Solaris 2.2 thread model?

Solaris 2.2 threads are scheduled in priority order and are preemptible. Priorities go from 0 to 128 in increasing order.

Is context saved per thread, or per light-weight process (i.e., where is the context saved, in kernel or in user space)?

Threads are user-level entities. Their context is saved and restored by the threads library. A thread executes on an LWP until it blocks on a synchronization object or is preempted by a higher priority thread. It is the threads library that manages scheduling threads to a pool LWPs. Typically there are more threads being scheduled onto a smaller pool of LWPs.

For which of the following thread functions are high-level APIs provided:

a. creation
b. termination
c. querying
d. scheduling
e. preemption
f. suspension/resumption
g. asynchronous activities

We support a through f. With threads, there is no reason to have a separate set of interfaces to support asynch activities.

Which of the following locking strategies are provided:

a. spin locks
b. barrier locks
c. semaphores
d. concurrent reads/exclusive writes
e. mutexes
f. condition variables

We support c through f.

What is the correspondence between user threads and kernel objects (processes or light-weight processes):

a. one user thread - one kernel object
b. many user threads - one kernel object
c. one user thread - many kernel objects
d. many user threads - many kernel objects

d. This is the model that Sun pioneered. It is usually referred to as a two-level threads model.

What thread-specific data is maintained by the Solaris 2.2 thread library?

The thread structure contains all the per thread data. This primarily is a stack, a set of registers, a priority, a thread ID, and a signal mask.

Describe signal handling in detail.

POSIX 1003.4a has adopted our signal handling model as described in draft 7.

Are signals between threads of the same process permitted? ...between threads of different processes?

Threads are not visible between processes. A thread can direct a signal at a thread within its process. A signal sent to a process can be handled by any thread within the process that has the signal unmasked.

How is the traditional process model supported, i.e., how is backwards compatibility provided?

A traditional process is simply a single threaded process.