SUN MICROSYSTEMS COMPUTER CORPORATION

SunOpsis

The Solaris 2.0 Migration Support Centre Newsletter

1-800-363-6200

VOLUME 1, NUMBER 2, SEPTEMBER 1992

The Publisher's Corner

Dean.Kemp@Canada.Sun.Com
Mgr Solaris 2.x Migration Support Centre

Welcome to the second edition of our newsletter. We hope you continue to find it interesting and useful. As part of our ongoing efforts we have two new support programs that will be available to a limited number of customers. If you are interested in either of these programs you should contact your local Sales Account Manager or Systems Engineer, so that you can be approved by your Area Transition Manager.

The first product we will be supporting is the Early Access of Solaris 2.1. As outlined in one of our articles in this issue, Solaris 2.1 has a number of interesting new features and most importantly supports all Sun Sparc platforms.

Our second service is the support of Early Access to User Level Threads (or the MT Tools as they are also referred to). These will allow software developers to create applications which execute more efficiently on a uniprocessor and that will also take advantage of our scalable MP architectures. This issue of the newsletter has an article on User Level Threads that you will want to read for more details. The additional approval of the Product Manager is required to participate in this program.

We are looking forward to your participation in both of these activities. Please do not hesitate to contact me if you have any questions or suggestions. We are here to make your transition as efficient as possible and value your input regarding our services.

The OPCOM T-Shirt Contest

Brian A. Onn, Editor-in-Chief

Would you like to win an OPCOM T-shirt? We have several that we will be giving away in our new contest. This contest is open to every OPCOM customer and Sun Microsystems employee.

We pondered over this one for a while, trying to decide just how we would offer it, and most importantly how we would select the winner. We wanted to be fair to everyone, and did not want to restrict the contest in any way, or inadvertantly rule anyone out with our choice of the contest details.

The rules are simple: write a feature article for SunOpsis, and email it to contest@opcom.canada.sun.com. If your article is selected for printing, you will win a t-shirt. Your article should be related to Solaris 2.x in some manner, and can be any reasonable length. Your article will become the property of Sun Microsystems Computer Corporation, and credit will be given to you as the author. If you cannot email us the article, you may fax it to us as well at 416-477-6483.

INDEX

The contest has no closing date yet, but we will choose one article from the ones we have received up to the 1st day of each month. That article will go into that month's newsletter, published mid-month.

Solaris 2.1 Features

Larry.Tsui@Canada.Sun.Com

Solaris 2.1 will be the next release after Solaris 2.0. It is the first release of Solaris that fully supports the multi-processor environment and it also provides performance improvements over Solaris 2.0. Solaris 2.1 is now available for early access. Here are some of the features that will be in Solaris 2.1:

- **Additional platform support:** Solaris 2.1 will run on sun4, sun4c and sun4m architectures.

- **Peripheral support:** Most of the peripherals will be supported by Solaris 2.1, including all disk types (SCSI, IPI and SMD). Some of the peripherals that will not be supported are 1/2" tape drive with the tapemaster controller, ALM-1, cg9 and TAAC.

- **Tagged Queuing SCSI driver:** Solaris 2.1 will supply a tagged queuing SCSI driver on supported platforms (SS10, etc) for faster SCSI performance. Tagged queuing allows multiple SCSI commands to be issued by the driver without waiting for status from drives.

- **Audio:** There will be support for 16-bit audio on the SS10.

- **Administration:** There will be three additional GUI (graphical user interfaces) for system administrators, namely Printer Manager, Software Management Tool and User Manager. The printer manager allows easy setup for both local and remote printing. The Software Management Tool is GUI interface to packages. The User Manager is an enhanced tool for user account management.

- **OpenWindows:** Solaris 2.1 will ship with OpenWindows 3.1, with extensions for dynamically loading new frame buffer drivers.

- **Localization:** Solaris 2.1 will be localized for major European languages (German, French, Swedish and Italian) as well as Japaanese There will be separate AnswerBook CDs, hardcopy documents (e.g. enduser kits, system accessory kits) for these localizations.

- **System Performance:** Solaris 2.1 provides much improved system performance over Solaris 2.0. There will be some bug fixes integrated into this release.

Preinstall Stubs: Preinstall stubs will allow easy customization of Solaris installation at a per-site level.

User Level Threads for Solaris 2.x

Srinath@Corp.Sun.Com

This article describes a model for multiple threads of control within a single process. The Solaris multi-threaded programming model has two concepts that must be understood by the programmer. The most important concept is the thread interface, which defines most aspects of the programming model. The second is the lightweight process (LWP) which is defined by the services the operating system must provide.

A thread of control is a sequence of instructions being executed in a program. For example, a traditional UNIX process has a single thread of control. A multi-threaded UNIX process is no longer a thread of control in itself, but is instead associated with one or more threads of execution. These threads execute independently, sharing process instructions and their global data. A change in shared data by one thread can be seen by the other threads in the process. For example, if one thread opens a file, another thread can read it. The thread architecture provides a variety of synchronization facilities to allow threads to cooperate in accessing shared data. These include mutual exclusion (mutex) locks, condition variables and semaphores.

Threads are implemented using LWPs. In the Solaris multithreaded architecture, a UNIX process consists mainly of an address space and a set of LWPs. Each LWP can be thought of as a virtual CPU available for executing threads. LWPs are separately dispatched by the kernel, and may perform system calls independent of each other, incur page faults and may even run in parallel on a multiprocessor architecture. When a thread needs to access a system service, take a page fault or interact with threads in other processes, it does so using the LWP that is executing it. The thread needing the system service remains bound to the LWP executing it until the system call is completed.

Threads are the primary interface for application parallelism. Few multi-threaded programs will use the LWP interface directly, as it is substantially more work than to use the threads library. Most programmers can use the threads interface and let the library take care of mapping threads onto the kernel primitives. For those programs that need to do so, direct control of LWPs is available.

A simple multi-threaded program used to update a bank balance is shown below. The program creates two threads, each of which execute the same function deposit() but with different arguments (the deposit amount). Note the use of mutual exclusion locks used to insure atomic updates of the variable bank_balance.

```
#include <thread.h>

void *deposit(void *);

int bank_balance;/* global variable */
mutex_t bb_mutex;/* protecting mutex */

int main(void)
{
    bank_balance= 200;
    thr_create (NULL, 0, deposit, 1000, 0, 0);
    thr_create (NULL, 0, deposit, 300, 0, 0);
    return(0);
}

void *deposit(void *args)
{
    int deposit = (int) args;

    mutex_lock (&bb_mutex);
    bank_balance = bank_balance + deposit;
    mutex_unlock (&bb_mutex);
}
```

To become an Early Access site for the User Level Threads library, please contact your local sales account manager or Systems Engineer for approval from the User Level Threads Product Manager.

Batch Processing on Solaris 2.0

Ron.Winacott@Canada.Sun.Com

Solaris 2.0 provides a mechanism that allows new scheduling classes to be added to the kernel, without re-coding and re-compiling it. The loadable scheduling classes that ship with Solaris are Time Share (TS) and the Real Time (RT) classes . This article describes an implementation of a BaTch (BT) scheduling class for Solaris 2.0. For the sake of simplicity, I will refer to a process as a single threaded process. This does not mean that BT will not work with multi-threaded programs (i.e. user level threads).

There are several important characteristics of Solaris 2.0 that make it possible to implement a new scheduling class like BT. UNIX Operating Systems that preceded Solaris 2.0 did not have these features.

- **Loadable scheduling classes** enabling a new class to be created and loaded into the kernel, and the kernel being able to call the class specific scheduling functions.[1]

- **Table driven global priorities**. Each class using a section of the global priorities from 0 to 159.

- **Class specific proc structures**. The kernel can have class specific data associated with each process that has meaning to that class only. This enables the developer to schedule processes differently within each class.

- **Class specific scheduling functions**. The kernel will call the class functions from a list of defined functions.

- **Ability to alter the dispatch table** of any scheduling class. The dispatch table is defined in a separate loadable module from the scheduling class, enabling the system programmers to alter the table to fit any new classes into the global priority scheme.

- **A fully pre-emptive kernel**, which allows a higher priority process to run when it needs to run, without waiting for the lower priority thread to give up the cpu.

The TS dispatch table starts at global priority 0. The first thing I had to do was move the TS class up, to make room for the BT class. Using the source code from the man page **ts_dptbl(4)**, I shortened the table by four slots (priorities). The new TS table starts at global priority four. This opened up the priorities zero to three for the BT class.

The Idle Loop

Each hardware cpu has a kernel idle thread that it runs if there are no runnable processes. The idle thread has a global priority of -1 to insure that it never runs other than at idle time. This is similar to the idle loop in the Solaris 1.x kernel. The difference here, however, is that the Solaris 2.x idle loop is truly an idle loop, that does no work. The Solaris 1.x idle loop has the additional task of processing the stream queues. This implies that the Solaris 1.x idle loop really has to run, otherwise the streams processing would stop. In the Solaris 2.x kernel this is NOT true anymore, and the idle thread does not need to run. Knowing this, I set up the BT class to run in the same way. Since the global priorities are from 0 to 3, processes in the BT class will only run if there are no runnable TS or RT processes. Also, the BT priorities are fixed, so it will never change due to the amount of cpu time the process has used, or the amount of time the process has to wait for the cpu.

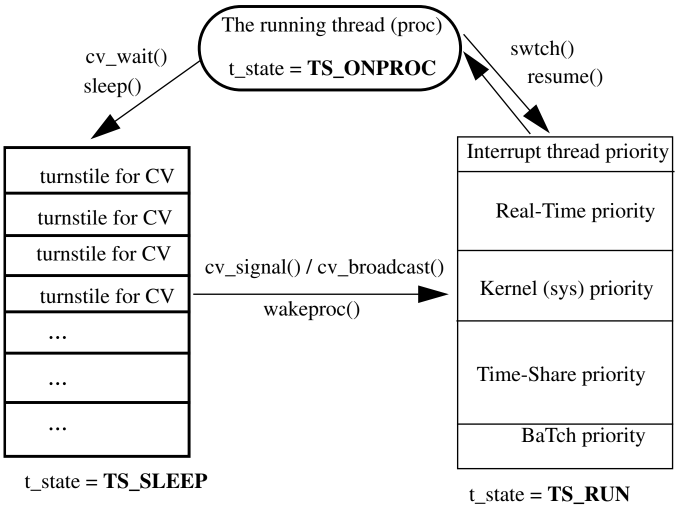
As stated above, the BT class has fixed priorities like the RT class, and a longer than normal time quantum. The time quantum is long because the running batch process should not surrender the cpu until it is done, or blocks. (i.e., waiting for I/O)[2]

You can start any number of threads with the same batch priority, and they will time slice among themselves whenever one blocks or expires its time quantum (assuming there are no other higher priority runnable process.) This time slicing on a dis-

1. See man modinfo(1), modload(1), modunload(1).

2. In the next release, I am thinking of dropping the idea of time quantum altogether. This would mean the BT process would never surrender the cpu until it blocks or is pre-empted.

patch queue is a problem for BT. If you want the historical notion of batch processing, you will want the second process to run **only** after the first one has exited. This is not possible with just the BT scheduling class because it has no control over when a process will block, and can't stop the next runnable process on the same dispatch queue from running (an article next month will describe a companion tool, `bt_tool`, that will provide this functionality) . The diagram below outlines the path a process will take going from the dispatch queue (TS_RUN), to running on a cpu (TS_ONPROC), and (if blocked) to the sleep queue (TS_SLEEP).



The running process may not always end up on the sleep queue. If it is pre-empted by a higher priority process, or if it uses up all of its time quantum, the process would be placed on the back of the dispatch queue. This is how it works in release 2.0 of BT. The next release may have the pre-empted batch process go to the front of the dispatch queue, not the back, and there may not be a time quantum. This could help simulate true batch processing.

The BT class defines the following functions to the kernel in the `classfuncs` struct (not all of them are listed here)

bt_enterclass() is called when a process enters the BT call to allocate a `bt_proc` structure and initialize it.
bt_exit() is called when a process in the BT class exits, so the kernel can free the memory used by the `bt_proc` structure.
bt_fork() is called by the system fork() to take care of the class specific data. It allocates a `bt_proc` struct for the child.
bt_preempt() is called when you are being pre-empted. This allows the BT class to put the still-runnable process back on the front of the dispatch queue.[1]
bt_sleep() is called if you are about to go to sleep. The BT class does very little here, other then set a state flag to indicate this thread was running.[2].

1. in contrast, the TS and RT classes would both put the process to the back of their queues to permit other processes on that queue to run

2. this is used in the bt_swapout() function

bt_wakeup() is called when the process is moved from the sleep queue to the dispatch queue. BT would put the process on the front of the dispatch priority queue.
bt_tick() is called every clock tick. It decreases the time quantum. At the next release of BT, this function may do nothing.

The other functions defined in the `classfuncs` structure are used to nominate a process to swap in or out, get and set the class specific data, get and set the class parameters (i.e. priority). Each of these functions are implemented in BT in a way that maintaines the batch processing semantics.

Using the BT class

After installing BT and replacing the TS_DPTBL module with the one I supply, your TS dispatch table would look like:

```
% dispadmin -g -c TS
# Time Sharing Dispatcher Configuration
RES=1000

# quantum tqexp slpret maxwait lwait   PRIORITY
    1000      0      9       5      9      #     0
    1000      0     10       5     10      #     1
    1000      1     11       5     11      #     2
    1000      1     12       5     12      #     3
    1000      2     13       5     13      #     4
    1000      2     14       5     14      #     5
     800      3     15       5     15      #     6
     800      3     16       5     16      #     7
     ...    ...    ...     ...    ...     ...   ...
     100     43     53       5     53      #    53
     100     44     54       5     54      #    54
     100     45     54       5     54      #    55
```

Note that the priority level starts at 0 on the output from the dispadmin command no matter what class you look at. The priority levels from the dispadmin command are an offset into the class specific dispatch table, and are added to the global priority where the class starts. That is, if the RT class starts at global priority 100, then RT priority 12 equals global priority 112. (The time quantum values may be different on your system, as I have changed them since this release.)

If you saved a copy of the output from the old TS dispatch table you will notice some changes. The table still looks like it is using priority 0 through 3, but in my TS_DPTBL module, I've set the starting point to priority 4. Remember, all the priority numbers in the above table are relative to the table only.

You can look at any of the scheduling class dispatch tables with the **dispadmin(1M)** command. With BT installed, dispadmin will show you the BT table as well:

```
% dispadmin -g -c BT
# Batch job Time Dispatcher Configuration
RES=1000

# TIME QUANTUM                        PRIORITY
# (bt_quantum)                          LEVEL
     10000                      #         0
```

```
 10000                              #        1
100000                              #        2
100000                              #        3
```

Generally, to start a command in a different scheduling class, you would use the **priocntl(1)** command. (**init(1M)** is started by default in the TS class so, all other commands inherit the parent class unless you change classes.) To start a command in BT you would enter:

```
% priocntl -e -c BT -p1 make -f Makefile.solaris
```

The `-e` is to execute the command, the *-c BT* is the class, the *-p1* is the class priority to run the command at, and the *make -f Makefile.solaris* is the command to run. The priocntl command can be used to look at all the processes in any class by entering:

```
% priocntl -d -i class BT
BATCH JOB PROCESSES:
    PID      BTPRI     TQNTM
   9369         0      1000
```

I supply a shell script called **bt** that will start any command in BT class at priority 0.

A good program to run in batch mode to see how much idle time there really is on your system is `/usr/openwin/demo/ico`.[1] This is a graphical program with a bouncing ball in a window. The ball will only move when there are no other runable RT or TS processes.

You will notice some strange things if you run a perfmeter at the same time you run ico. The cpu will go to 100%, which is fine. This simply means there is no idle time on your system, which is true, since ico is using all of it. The other thing you will notice is the context switching count will increase. ico is being preempted each time there is an interrupt or one of the other processes on your system becomes runnable. If you are running OpenWindows, this would be any time you do anything, such as move your mouse.

In practice, I have found the BT class to be very useful. I often use it to start a large make (like xview 3.0) in the background. This leaves the system very responsive while the make is running. and any child of make (like cc) inherits the scheduling class of the parent, so they also run in the BT class.

How to Get It

The BT scheduling class is available electronically via the OPCOM FTP server, under the tar directory (See "The OPCOM FTP Server" on page 11.) Please report bugs, change requests and general comments to me, as I am interested in hearing what uses you have for BT.

1. this file is only available when the SUNWowdem package is installed

NFS Interoperability

Georg.Nikodym@Canada.Sun.Com

Most people simply use NFS without incident and take for granted that it just works. This article outlines two interoperability issues that have appeared in our experiences with Solaris 2.0. Keep in mind that these issues have come to light not from any deficiencies in NFS, but rather as a by-product of the growth and popularity that NFS has seen.

Older NFS Implementations

The first issue arose using some NFS implementations from other vendors. When one attempts to mount a filesystem from such a system, the mount may fail with a message to the effect:

```
nfs mount: server:/export/filesystem server not
responding: RPC: Authentication error; why =
Invalid client credential
```

In attempting to diagnose such problems, the common suggestion is to check the remote mount daemon. There are some old mountd's that need to be started with a -n option to allow them to painlessly exist in current networks.

Unfortunately, in this case, the root of the problem was somewhat more difficult to unearth. A network analysis tool is indispensible in such situations and the **snoop(1M)** utility fit the bill perfectly. By running:

```
snoop -o outputfile host1 host
```

during the NFS mount attempt we were able to record all packets traversing the network between the two hosts in question. Subsequent analysis of this data showed that the mount request was being rejected by the RPC layer, not the server's mount daemon. That is to say, the server's mount daemon did not even see a mount request much less reject one.

Thanks to snoop, we knew that the problem lay in the RPC layer and we knew the exact rejection condition so it was time to dive into the RPC source code (which is publicly available, so you can too). In turns out that there are about five related reasons that the RPC layer would reject a request with this return code. At this point we had two suspects, the first being that the expanded UID/GIDs of SVR4 (historically the `uid_t` and `gid_t` were unsigned shorts, whereas SVR4 made them unsigned ints) were not being decoded correctly by the server. The second possibility was that there was some problem with the list of supplementary group IDs and the way they are decoded. Since we did not have access to the source code of the the server implementation (on the non Sun machine), in this case, we could only guess at what it looked like. So, as a parallel effort, I got the public RPC code for versions 4.0 and 3.9 (in the event that there were differences from Sun's own code base).

Fortuitously, there came into my possession a very old NFS design spec which among other things had an outline of XDR[1] which stated that the fundamental XDR type was an int (32-bit quantity). This fact ruled out the first theory and did serious damage to the second. My examination of the old 3.9 RPC code was, on the surface, fruitless as well, since the files I was concerned with had not changed a great deal between releases. In desperation, I did a brute force comparison of every file in the RPC distributions I had. While expressing my frustration to a co-worker, up popped the answer on my screen.

It turns out that the supplementary group list has a limit on the number of groups one may be in. In RPC3.9 it was 8, and in 4.0 it became 16 (defined by NGRPS in `rpc/auth_sys.h`). When the credentials for a mount request are being constructed, the list is filled (up to the limit defined by NGRPS) with the GIDs of the groups that the requesting user (usually root) is a member of. The `/etc/group` file supplied with Solaris has root listed in 10 groups, which violates the group membership limit on systems based on RPC3.9.

The simple work around is to edit the `/etc/group` file to remove root from membership in the numerous groups that this account is listed in. This change will not affect operation of the system since the root account is all powerful anyway.

Making Special Files

The second issue involves the creation of device nodes on an NFS filesystem (ie. with the **mknod(1)** command). A device node in the filesystem is seen as a special file with an associated major and minor number. What happens behind the scenes is that the major and minor numbers are "pasted" together into a single `dev_t` (the major number is shifted left and the result is ORed with the minor number). Under SVR4, the fundamental system types have been expanded (as a concept, this is known as EFT). In particular, the `dev_t` type has gone from a `short` (16 bit value) to an `int` (32 bit value) and thus the number of bits to shift the major number to derive a `dev_t` has changed.

Current NFS implementations are based on a specification that did not directly define how the characteristics of a special file were to be transmitted over the wire during an NFS CREATE operation. So what is done is that the file creation request goes out with the `dev_t` stuffed into the size field (the fact that you're creating a special file is known from the mode bits).

Now when creating a special file on an older server from an SVR4 client, the **mknod(1)** commfand constructs a 32 bit `dev_t` which gets decoded by the client as a 16 bit `dev_t` so the extracted major number is usually zero thus resulting in an incorrect special file being created on the server.

1. eXternal Data Representation - a protocol for ensuring that binary data is moved around in an architecturally neutral manner over the network - XDR is fundamental to the success of NFS in multi-vendor networks

Interestingly, however, the **mknod(1B)** command from the Berkeley compatibility library constructs a 16 bit `dev_t` which works correctly with devices whose major/minor numbers are less than 255 on both the client and server.

Now, before you start worrying about using your Solaris2.x machine as a server, rest assured that the NFS implementation is aware of both kinds of `dev_t`'s and will do the right thing.

Migrating from SunOS 4.1.x to Solaris 2.0

Gerry.Singleton@Canada.Sun.Com

Migrating from one operating system to another that is completely different presents many pitfalls. This article will attempt to address the major issues of this type of migration.

The first step is **PLANNING**. This means sitting down, examining system files such as `/etc/passwd`, and evaluating their use under the new operating system. As a guide, there's a list of important files at the end of this article. There will be others that you will select for your own installation.

Once you have your list of important files, you should back them up, either copying them to a local disk that will not be affected by the installation or by using removable media, such as tape or floppy disk.

Copying to local disk offers easy access for comparison and modelling as you prepare your newly installed system. In the case of floppies, you should have sufficient space for those files listed below but their 1.2Mb capacity may present limitations. Now you can install Solaris 2.0.

Once the installation is completed, you can begin the process of molding your system to suit your working environment, using the previously saved files for reference. **Do not copy these directly over top of the existing Solaris files, since they are of a different format.**

For the purpose of this article, we will restrict the discussion to some of the more important system files. The handling of other files will be similar.

Setting up mount points

Solaris 2.0 uses `/etc/vfstab` for describing filesytems that you would like to mount at boot. The Solaris 1.x counterpart is `/etc/fstab`. You will use your copy of your old `/etc/fstab` in updating and correcting your new `/etc/vfstab`. Once you have finished editing `/etc/vfstab` and are satisfied with its construction you may chose to mount everything by entering the command `mountall -a`. Here's a partial list of the many manpages and documents you should read for explanations and descriptions of /etc/vfstab, mounting and other administrative issues: **vfstab(4)**; **mount(1)**; **mountall(1m)**; Routine System Administration Guide - SunOS 5.0; Administering NFS and RFS - SunOS 5.0; System Transition Guide for Users - SunOS 5.0

vfstab/fstab differences

The format of an fstab entry in SunOS 4.X was:

```
filesystem directory type options freq pass
```

The format of an vfstab entry in SunOS 5.0 is:

```
device   device   mount FS   fsck mount   mount
to mount to fsck  point type pass at boot options
```

Exporting filesystems in Solaris 2.0

Once /etc/vfstab has been customized to reflect your local filesystems, its likely you'll want to provide access to others. Doing so under Solaris 2.0 is somewhat different than under Solaris 1.x. The method is similar, but not the same, and you'll need the saved copy of the /etc/exports for reference.

Files are exported in Solaris 1.x via the **exportfs(8)** command and the /etc/export file. The corresponding Solaris 2.0 capability is implemented using the **share(1M)** command and the /etc/dfs/dfstab file.

Using the saved /etc/exports file as a model, perform the following steps:

1) Edit /etc/dfs/dfstab to export files. You may refer to **dfstab(4)**, which describes share commands that are executed upon initialization. This is similar to a series of fields in your file /etc/export. The command syntax used in /etc/dfstab is the syntax for **share(1M)**.

2) Since you've just installed, nothing is shared yet. Before you reboot, touch /etc/dfs/sharetab. This action sets things up so that /etc/rc3.d/S15nfs.server executes during boot up. If you're in a hurry you can start things by entering /etc/rc3.d/S15nfs.server start on the command line. By using this approach, you ensure that rpc.mountd and the nfsd daemons are running. Either way you, you will have a system that exports files.

The Automounter

Next you should set up the automounter. Automount is a system daemon that is automatically invoked at boot time. It transparently mounts NFS file systems as they are needed, and unmounts them when they are no longer required.

There are some differences between the Solaris 1.x and Solaris 2.0 versions of the automounter. You must address these before you can establish an operational environment similar to the one you had before the migration. One difference, for example, is the -m command line option. The Solaris 1.x automounter uses this option to prevent reading the auto.master NIS database. Under Solaris 2.0, this option has been eliminated. Some files are renamed, as well. The Solaris 2.0 automounter uses /etc/auto_master and /etc/auto_home files, while the Solaris 1.x equivalents are /auto.master and /etc/auto.-home (Note: the manual page for automount has been moved from section 8 to section 1m.)

Device Names

Even with Binary Compatibility Package installed, you should be aware of proper device naming under Solaris 2.0. Previously, naming conventions used the form /dev/sdXY, where X is the disk number (e.g., 0 through 4) and Y is the disk partition name (e.g., a through h). The new format uses /dev/dsk/c0tXd0sY, where X is still the disk number (0 through 4); but Y is now the disk partition number (0 through 7) that corresponds to the old letter designation. For instance, the 'h' disk partition in the third SCSI disk becomes /dev/dsk/c0t2d0s7.

Using this convention is important for a typical /etc/vfstab entry. For example, /etc/vfstab must contain lines in the following format:

```
#device   device  mount FS  fsck mount   mount
#to mountto fsck point typepass at bootoptions
#
/dev/dsk/c0t3d0s0/dev/rdsk/c0t3d0s0/ufs1no-
/dev/dsk/c0t3d0s6/dev/rdsk/c0t3d0s6/usrufs2no-
/dev/dsk/c0t3d0s7/dev/rdsk/c0t3d0s7/export/
homeufs3  yes -
/dev/dsk/c0t3d0s5 /dev/rdsk/c0t3d0s5 /opt ufs 4
yes -
```

Recommended Reading

- vfstab(4) - manual page
- Routine System Administration Guide - SunOS 5.0
- Administering NFS and RFS - SunOS 5.0
- System Transition Guide for Users - SunOS 5.0
- mount(1) - manual page
- automount(1m): the manual page for automount
- Administering NFS and RFS - SunOS 5.0
- Routine System Administration Guide - SunOS 5.0

Files to Preserve for Migration

aliases	fbtab	filetype	format.dat
gettytab	group	holidays	hosts
inetd.conf	magic	netmasks	networks
passwd	printcap	protocols	rc
rc.boot	rc.ip	rc.local	rc.single
remote	sendmail.cf	services	termcap
ttytab			

SunOS 5.0 - the loadable operating system.

Richard.Marejka@Canada.Sun.Com

With the introduction of Solaris 2.0 Sun has implemented a kernel that does not require re-compilation to be modified, extended or tuned. This singular feature will enable better performance and easy addition of Sun and third-party software and hardware. What used be to the kernel under SunOS 4.X (`/vmunix`) is now implemented as a *core kernel* (`/kernel/unix`) and approximately 98 separate loadable modules. The core includes the VM-system, VFS/VNODE layer, system call interface and loadable module interface. All other features are contained in the loadable modules. These modules are only loaded into memory as required and are removed when the kernel detects a memory short-fall. Tuning is achieved with the use of the `/etc/system` file to set kernel parameters (more on this in a later article).

Loadable Modules

The idea of loadable modules is not new to SunOS. The 386i system was the first to implement the feature. With the introduction of the SPARCstation1, SunOS 4.0.3c included support for loadable modules. This feature has spread to later versions of SunOS 4.X with both loadable drivers and system calls.

As mentioned above, the Solaris kernel is completely loadable. This means that all "non-core" features of the operating system must be loadable. The system implements the following loadable types (from `/usr/include/sys/modctl.h`):

- drivers— arp, fd, sd, le, cgsix, etc.
- system calls— msgsys, nfs, rfs, semsys, shmsys, etc.
- file systems— nfs, procfs, specfs, tmpfs, etc.
- streams modules— arp, ms, kb, ldterm, etc.
- scheduling classes— TS, RT, etc.
- exec file types— aoutexec, elfexec, intpexec.

Documentation is supplied for drivers and streams modules in AnswerBook, various SMCC Education courses and printed material. Of the remaining four classes of modules there has been interest expressed in both system calls and scheduling classes. The remainder of this article will describe the loadable system call interface (a companion article will describe an implementation of a Batch Scheduling Class for Solaris).

Loadable System Calls

System calls are the interface between user and kernel. The system call interface provides an application with access to kernel resources and services. Solaris supports 146 different calls as documented in **List(2)**.

A loadable system call implements a system call that is not part of the *core kernel*. Solaris comes with a small number of loadable system calls, encompassing System V IPC, accounting, NFS and RFS (look in `/kernel/sys` and `/usr/kernel/sys`). All of the other system calls (i.e. **open(2)**, **close(2)**, **gettimeofday(2)** etc) are implemented in the "core".

A loadable system call can be used to provide special access to a streams module or hardware device. This type of access would be similar to the "out-of-band" concept on a TCP/IP connection.

With a little effort and some investigation the necessary pieces of information to port an existing loadable system call or write a new system call can be assembled.

Starting with the modlinkage structure in `/usr/include/sys/modctl.h`:

```
...
struct modlsys {
 struct mod_ops *sys_modops;
 char *sys_linkinfo;
 struct sysent *sys_sysent;
};
...
struct modlinkage {
 int ml_rev;/*rev of loadable modules system*/
 void *ml_linkage[4]; /*
 * NULL terminated list of
 * linkage structures*/
};
...
```

where `ml_linkage[0]` is a pointer to a `modlsys` structure. Within the `modlsys` structure the `sys_linkinfo` member is a string used with the **modinfo(1M)** command. The `sysent` structure is found in `/usr/include/sys/systm.h`:

```
extern struct sysent {
    char sy_narg; /* total number of arguments */
     char sy_flags; /* various flags as defined
below */
     int (*sy_call)(); /* handler */
```

```
     krwlock_t * sy_lock; /* lock for loadable
system calls */
    } sysent[];
    ...
```

the members `sy_flags` and `sy_lock` can both be `NULL`. The `sys_modops` member of a `modlsys` structure requires some research. Answerbook documents that the `mod_opst` member for a driver is `mod_driverops` and for a streams module is `mod_strmodops`. Checking `/kernel/unix` for like modules, the following symbols are found:

```
[1538] 0xf00a5e1812  OBJT GLOB 0 4 mod_schedops
[1655] 0xf00a5df412  OBJT GLOB 0 4 mod_syscallops
[1740] 0xf00a5e0012  OBJT GLOB 0 4 mod_fsops
[3279] 0xf00a5de812  OBJT GLOB 0 4 mod_driverops
[3638] 0xf00a5ddc12  OBJT GLOB 0 4 mod_miscops
[3649] 0xf00a5e2412  OBJT GLOB 0 4 mod_execops
[3788] 0xf00a5e0c12  OBJT GLOB 0 4 mod_strmodops
```

Using symbol [1655] completes the exercise. The skeleton section below assembles all these points into "boiler-plate" form. This should be sufficient to write a loadable system call. The new manual section (9) documents the functions, structure and entry points (for drivers) in the kernel. To guarantee operation only use these functions. The usage of any other functions will not assure operation in future versions of Solaris.

To compile your system call and install it requires five steps. They are briefly:

1) Compile. Use the ANSI C compile flag to assure maximum

2) type checking
   # cc -Xc -c -D_KERNEL mysyscall.c

3) Link
   # ld -r -o mysyscall mysyscall.o

4) Install
   # cp mysyscall /usr/kernel/sys/mysyscall
   # chmod 0755 /usr/kernel/sys/mysyscall
   # chown root /usr/kernel/sys/mysyscall
   # chgrp sys /usr/kernel/sys/mysyscall

5) Add system call to `/etc/name_to_sysnum`
   You must add a system call entry to this file. An entry looks like:

   ```
   mysyscall N
   ```

   where N is a number less than the kernel symbol sysentsize (186 in Solaris 2.0 FCS) and not currently in use. It is VERY important that you do not choose a currently defined system call number. The name "mysyscall" must match with the name of the file installed in .

6) Reboot. This will cause the kernel to re-read the `/etc/name_to_sysnum` and build a reference to the system call.

   ```
   # reboot
   ```

Skeleton

The small portion of code below documents the "front-end" of a loadable system call.

```
/*
 * Include Files
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/errno.h>
#include <sys/modctl.h>

/*
 * Constants & Macros
 */

#defineNUMBER_OF_ARGS3

/*
 * Data Declarations
 */

struct MySystemCallArgs {/* For Example*/
    int command;
    void*buffer;
    size_t bufsiz;
};

/*
 * External References
 */

extern struct mod_opsmod_syscallops;
externint MySystemCall(struct
MySystemCallArgs*,
                        rval_t *rvp );
/*
 * External Declarations
 */

static struct sysent MySysent = {
    NUMBER_OF_ARGS,
    0,
    MySystemCall,
    0,
};

static struct modlsys modlsys = {
    &mod_syscallops,
    "MySystemCall - for modinfo",
    MySysent,
};

static struct modlinkage modlinkage = {
    MODREV_1,
    { (void *) modlsys, NULL, NULL, NULL }
};
```

```
/*
 * Loading Interface.
 */

int
_init() {
    return( mod_install( &modlinkage ) );
}

int
_fini(){
    return( mod_remove ( &modlinkage ) );
}

int
_info( struct modinfo *mp ) {
    return( mod_info( &modlinkage, mp ));
}

/*
 * MYSYSTEMCALL
 *
 */

int
MySystemCall( MySystemCallArgs *ap,rval_t *rvp
) {
    int status= 0;
    ...
    return( status );
}
```

More Information

Our FTP server has a sample loadable system call that you can download. (See "The OPCOM FTP Server" on page 11.) The system call implements a kernel panic as a system call accessible from a user application.

SOLARIS 2.0 HINTS AND TIPS

The entries in this section are a series of interesting items discovered by users of Solaris 2.0.

To submit an entry, send email to opcom@sun.com with an appropriate subject line.

Dual Boot

David.Guerra@Canada.Sun.Com

Some customers like to have both Solaris 1.x and Solaris 2.0 on the same system and be able to boot from either. If the system is already running Solaris 1.x and you are loading Solaris 2.0 on a different disk, suninstall could repartition all your disks for you.

In the disk config section of suninstall, the first disk you must look at is the disk that Solaris 2.0 will be installed on. This will automatically partition this drive for `/, /usr, /opt a/, /usr, /opt and /export/home`. If you look at the second drive (the disk that contains Solaris 1.x) suninstall will configure it to predefined sizes. At this time choose option 'a' (initial partitioning) and hit the space bar to change to 'existing'. This will bring up the existing label on that disk. Now remove any references to mount points using option 'd'. This will prevent suninstall from configuring those partitions. When exiting the disk form, suninstall will complain that some disk space is not used; you can ignore this.

Continue with the suninstall on the first disk, and your existing installation on the second disk is safe from harm.

Loading Dependant Drivers

Ron.Winacott@Canada.Sun.Com

If you are writing a loadable driver that depends on some other driver being loaded first, you can use the `_depends_on[]` array to ensure that the other driver is loaded before yours is. For example, a line like:

```
static char _depends_on[]="misc/des";
```

in your driver tells the kernel run-time linker that your module depends on the `misc/des` module.

Lock Daemon Warnings

David.Guerra@Canada.Sun.Com

Nfs mounting file systems from a Solaris 1.0 system to a Solaris 2.0 system produces the following error:

```
May 21 16:43:30 scirocco automount[146]: No
network locking on dolphin : contact admin to
install server change
```

This is because the lockd on the server is too old for the Solaris 2.0 locking protocol. The 100075 lockd patch must be installed on the Solaris 1.0 server. This patch is available from our FTP server (See "The OPCOM FTP Server" on page 11.)

Failing Driver Attaches

Brian.Onn@Canada.Sun.Com

Be aware of the distinction between nodev() and nulldev(). The function nodev() always fails, and nulldev() always succeeds. If your driver has put nodev() in the cb_ops or dev_ops structures for functions that you don't support (i.e. probe), your driver may not get loaded since a required function failed. Use nulldev() instead in this case.

Driver Config Files

Brian.Onn@Canada.Sun.Com

Need a quick way to get specialized configuration data to your driver? You can do this with the xx.conf file and the **ddi_getprop(9f)** and **ddi_getlongprop(9f)** functions.

Create a name-value pair in your driver's xx.conf file and your driver can easily read the value of the named property. There is no special meaning assigned to the properties defined in the xx.conf file, so you're free to use this mechanism to pass information to your driver. Note that the xx.conf file is only read when the driver is loaded.

Cron Log Files

Georg.Nikodym@Canada.Sun.Com

There is a potential for the cron log files `/var/cron/log` to grow without bounds. This is because the file `/etc/cron.d/logchecker` checks root's ulimit to determine the size of the disk file. If you desire a fixed limit in disk blocks for cron logs, change line 34 in this script from

```
exit
```

to

```
LIMIT=nnnn
```

where nnnn is the desired maximum number of disk blocks.

The OPCOM FTP Server

The OPCOM FTP server is available 24 hours a day, and contains programs that have been ported to Solaris 2.0, documentation, selected white papers, RFC's, etc. Many of the example programs presented in the text of this newsletter can be downloaded from our FTP server.

The server is located internally to Sun Microsystems of Canada, protected behind Sun's Internet firewall, and is mirrored to our Internet host **opcom.sun.ca**. If you are not directly connected to the Intenet, or otherwise cannot FTP to our server, you can still access the files we provide. We have set up an automated email based file server for this purpose. To find out how you can get files from this server, send email to **ftp@opcom.canada.sun.com** (or ...!uunet!sun!suncan!opcom!ftp), with the subject **help**.

If you are internal to Sun Microsystems, Inc., you can access the server via FTP to opcom.canada. If you are outside Sun, please use the server at opcom.sun.ca. Note that the opcom.sun.ca link is noticeably slow.

Calling Solaris 2.0 Support Dispatch

Lisa Thompson / Deborah Hummel

We can be reached at **1-800-363-6200** or email:

solaris2@sun.com - for Solaris 2.x issues
threads@sun.com - for User Level Threads issues

Your call will be answered by one of our dispatchers. Information obtained will then be used to open up a service call on your behalf and will be forwarded to the appropriate engineer. The engineer will then respond within the standard 1 hour call back time. Email response is much the same, however an email will be returned to you by the Dispatcher letting you know that the call has been logged and giving you the Call # for your information.

OPCOM Registration

Lisa Thompson / Deborah Hummel

How is support determined and how do I get there from here?

Customers who qualify for support services is determined by Solaris 2.0 Migration Managers.Your SUN Sales Representative will correspond with management staff and forward the necessary information required for registration.This registration is then sent to OPCOM for verification purposes when customers call for support.

When there are revisions, deletions or additions to be made to registrations which are currently in the data base you must contact the dispatchers at OPCOM or send email with your changes to:

solaris2-register@sun.com

Please keep in mind this includes stand-ins for vacation coverage.

Future SunOpsis Articles

Porting NIT to DLPI
Driver Debugging
Porting Experience: X11R5
IP Multicasting
Batch Scheduling - part II
NIS+ Programming Interface
User Level Threads - an example
POSIX compliance
Porting Device Drivers
and still more Hints & Tips

SunOpsis is published monthly by OPCOM.

To subscribe or contribute articles please contact:

Solaris 2.0 Support Centre
100 Renfrew Drive,
Markham, Ontario,Canada
L3R 9R6

1-800-363-6200 or email us at
solaris2@Sun.Com

Publisher: Dean Kemp
Editors: Brian A. Onn / Gil Hauer
Design & Layout: Lisa Thompson / Deborah Hummel
Contributing Writers:
Enrico Giuditti
David Guerra
Gil Hauer
Deborah Hummel
Dean Kemp
Richard Marejka
Georg Nikodym
Brian A. Onn
G. Roderick Singleton
Lisa Thompson
Larry Tsui
Srinath Venkat
Ron Winacott