Calling Solaris 2.x 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?

Customer qualification for support services is determined by the Area Solaris 2 Migration Manager.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
User Level Thread Examples
NIS+ Programming
The Solaris 2.x Networking Model
The Transport Layer Interface (TLI)
Streams Programming
Security Features in Solaris 2.x
and still more Hints & Tips

SunOpsis is published monthly by
The Solaris 2.x Migration Support Center

To subscribe or contribute articles please contact:

Solaris 2.x Migration Support Center
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
Ed Milstein
Georg Nikodym
Brian A. Onn
John Richardson
G. Roderick Singleton
Lisa Thompson
Larry Tsui
Srinath Venkat
Ron Winacott

Solaris 2.1 Beta Boot Messages

Dave.Guerra@Canada.Sun.Com

When booting up the Solaris 2.1 beta CD the following warnings may be displayed. I have included some information regarding these messages. Each of these warnings may appear after the initial banner, printed below:

```
SunOs Release 5.1 Version beta_1.0 [UNIX(R) System V Release 4.0]
Copyright (c) 1983-1992, Sun Microsystems, Inc.
```

WARNING: DOWN-REV ROSS 605 CHIPS - PLEASE UPGRADE TO VERSION A.8

This tells you that the CPU modules in your 600MP system are below the required revision. Solaris 2.1 requires these modules to be at revision A.8. You should place a hardware service call and request an upgrade to your 370-1388 cpu modules. When placing the call, tell the dispatcher that this is Field Change Order number A0026-1.

WARNING: Cannot open /etc/path_to_inst

This message can be ignored when booting from the beta 1.0 CD, but is a concern when booting from your local disk. This warning is not present in the FCS CD release.

WARNING: clock gained xx days -- CHECK AND RESET THE DATE!

This warning can be ignored and is just a discrepancy between the date on the CD and the system date.

BT Scheduling Class problems

Ron.Winacott@Canada.Sun.Com

There have been some reported problems with the BT scheduling class, available on the OPCOM FTP server.

1) When the *install_BT* script is run, it will ask if you want the inetd set up to auto-start the rpc.btd. If you answer *yes* to this question, BT will not run, since rpc.btd is not installed properly into the inetd.conf file. You should answer *no* to the install question, and start the /var/bt/rpc.btd by hand as root.

2) The **bt** and **bt_tool** can't start a local command. The command must be found in your search path (excluding ".") to run. There is a problem with building the full path name when you try to run a command line such as *./command*.

Both of the above problems are being worked on and should be fixed in release 3.0.1 of the BT scheduling class.

X11R5 For Solaris 2.x

Larry.Tsui@Canada.Sun.Com

The X11R5 port to Solaris 2.x has been updated to support the cg3 framebuffer and multi-head display. You can get it via anonymous ftp from the OPCOM FTP Servers. Please refer to the file /pub/x11r5/README.x11r5 for more details.

Network Analysis

Brian.Onn@Canada.Sun.Com

Troubleshooting your network? Looking for rpc.etherd, or etherfind? You won't find them on Solaris 2.x. Instead, you *will* find **snoop(1m)** a superior and easier to use network monitor. The snoop command can save network traffic to a file for later analysis, it can give short one line traffic summaries, or overwhelm you with a complete breakdown of every protocol layer in a captured packet. The snoop program is very complete in its capabilities, so see the man pages for more information.

The OPCOM FTP Server

The OPCOM FTP server is available 24 hours a day, and contains programs that have been ported to Solaris 2.x, 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 Internet, 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.

The numbers will be different on your system, but the method will be the same. Now enter the *dispthread* address and the thread macro like **ff62d600$<thread** and press return. When you see the *cldata* field go by, you can press control-C to stop kadb from scrolling the information off the screen.

```
kadb[0]: ff62d600$<thread
0xff62d600: link     stk        stksize    affinity
            0        f02af930   1930       ffffffff
0xff62d610: affinitycnt flag    schedflag  state
            0           0       1          3
0xff62d620: pri         epri    pc         sp
            100         0       f0022b90   f02af8b8
0xff62d62c: wchan0      wchan   cid        clfuncs
            0           0       1          ff0e9544
0xff62d63c: cldata      ctx     lofault    onfault
            ff61ffc0    0       f000730c   0
^C
```

Now you have the priority of the thread (process) and the address in the kernel where that priority is stored.

The RT class has fixed priorities, so we can change it to a very low value and the scheduling class will not change it back on us.

Take the address on the left hand side of the display from the line with *pri* on it and enter it into kadb with the /d modifier. Kadb will display the decimal short int value of that address. If you did not change the priority of the test RT process from the default, it should equal 100. Now enter the same address again, but this time use the /w modifier (lower case "w") followed by a space and a "5". After pressing return, the thread's new priority is 0x5 hex. Now enter ":c" to continue to run the kernel. The runaway process is now running at a low priority of 5 and you can now ^C or terminate it with a signal, since any other process can preempt it.

This is how it would look:

```
kadb[0]: 0xff62d620/d
0xff62d620: 100
kadb[0]: 0xff62d620/w 5
0xff62d620: 0x64 = 0x5
kadb[0]: :c
```

SOLARIS 2.x HINTS AND TIPS

The entries in this section are a series of interesting items discovered by users of Solaris 2.x.

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

Reading Serial Line Modem Signals

Ron.Winacott@Canada.Sun.Com

The ioctl **TIOCMGET** was used in the past to read the serial line modem control status. Under Solaris 1.x, ioctl(fd,TIOCMGET,&rtn) would return the modem control line states in the int *rtn*. Under Solaris 2.x, you must use the I_STR ioctl to pass this ioctl to the serial driver. The format is:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <termio.h>
#include <sys/termios.h>
#include <errno.h>
#include <sys/tty.h>
#include <stropts.h>

get_modem_control(int fd)
{
    struct strioctl st;
    int     rtn;

    st.ic_cmd = TIOCMGET;
    st.ic_timout = 0; /* use the default 15 seconds */
    st.ic_dp = (char *)&rtn;
    st.ic_len = sizeof(rtn);
    if (ioctl(fd,I_STR,&st) < 0)
        perror("I_STR");
        exit(1);
    }
    printf("modem control = %d\n",rtn);
}
```

Pitfall When Merging AnswerBooks

Dave.Guerra@Canada.Sun.Com

When merging AnswerBooks, don't use a collection name longer than 10 characters. If you do, the search facility cannot be initialized.

The *abmerge* command is in the _data directory of the AnswerBook installation. For example, in the Solaris 2.1 Beta AB it is in `/opt/Solaris_2.1_AB/_data/SUNWab2_3`. To merge two or more AnswerBooks, change to this directory and type:

```
# ./abmerge
```

This will lead you through an interactive session to do the merge. When you are asked to enter the name of the merged AnswerBook, enter a name no longer than 10 characters. Once the merge is completed, you can access the contents of all the AnswerBooks you just merged from one navigator window.

```
        fprintf(stderr,"Can't get RT class info\n");
        perror("priocntl info");
        exit(1);
    }
    pcparms.pc_cid = pcinfo.pc_cid;
    ((rtparms_t *)pcparms.pc_clparms)->rt_pri = 59;;
    ((rtparms_t *)pcparms.pc_clparms)->rt_tqsecs = 0;
    ((rtparms_t *)pcparms.pc_clparms)->rt_tqnsecs = RT_TQINF;
    if (priocntl(P_PID,P_MYID,PC_SETPARMS,(caddr_t)&pcparms)
            == -1) {
        fprintf(stderr,"Can't change to RT class\n");
        perror("priocntl switch");
        exit(1);
        }
    /*
    ** setup itimer to check /proc every 20 seconds
    */
    sigemptyset(&sigact.sa_mask);
    sigact.sa_handler = sigtimeout;
    sigaction(SIGALRM,&sigact,NULL);
    it.it_interval.tv_sec = 20;
    it.it_interval.tv_usec = 0;
    it.it_value.tv_sec = 20;
    it.it_value.tv_usec = 0;
    if (setitimer(ITIMER_REAL,&it, NULL) == -1) {
        perror("Setitimer");
        exit(1);
    }
    while (1) {
        pause(); /* wait for the sigalarm */
    }
 }


void
sigtimeout(int sig)
{
    /*
    ** check for running realtime procs, that have more the
    ** maxsec time on them.
    */
    DIR         *dirp;
    struct      dirent  *dentp;
    char        pname[1024];
    int         procfd;
    int         saverr;
    prpsinfo_t  psinfo;

    if (( dirp = opendir("/proc")) == NULL ) {
        perror("rt_wd: open /proc");
        exit(1);
    }
    while (( dentp = readdir(dirp)) != NULL ) {
        if (dentp->d_name[0] == '.')/* skip . and .. */
            continue;
        sprintf(pname,"/proc/%s",dentp->d_name);
        retry:
        if (( procfd = open(pname,O_RDONLY)) < 0) {
            perror("rt_wd: proc open");
            closedir(dirp);
            exit(1);
        }
        if (ioctl(procfd,PIOCPSINFO, (char *)&psinfo) == -1
            saverr = errno;
            close(procfd);
            if (saverr == EAGAIN)
                goto retry;
            /*
            ** error
            */
            errno = saverr;
            perror("rt_wd: ioctl: PIOCPSINFO");
            exit(1);
        }
        close(procfd);
        /*
        ** if you get here, you have the proc info
        */
        if (strcmp(psinfo.pr_clname,"RT") != 0)
            continue;   /* not real time */
        /*
        ** Found a realtime proc ! Check the time.
        */
        usedsec = psinfo.pr_time.tv_sec;
        if (usedsec >= maxsec) {
            kill_pid(psinfo.pr_pid);
            continue;
        }
        /*
        ** Tell the ues he is getting close to being killed
        */
        if (usedsec > (maxsec / 2)) {
            fprintf(stderr,"rt_wd: Pid %d is at 50%% time
                or more!\n", psinfo.pr_pid);
        }
    }
    closedir(dirp);
}

kill_pid(pid_t pid)
{
    printf("rt_wd: killing pid %d\n",pid);
    if (kill(pid,SIGKILL)){
        perror("rt_wd: kill");
    }
}
```

The watchdog program is available from the OPCOM FTP Servers.

Using Kadb To Stop A Runaway Process

If your system was booted with kadb ("boot kadb" or change the *boot-from* environment variable in the boot prom to "kadb") you can press L1-A and the system will stop in kadb, the kernel debugger. At this point, you will have all the adb macros and commands at your disposal, with some minor differences (see man **kadb(1)**). To continue to run after stopping, enter ":c" (no quotes).

Having kadb at your disposal greatly simplifies things when it comes to runaway RT processes. Now when the system hangs, you can press L1-A to stop the system. You will see a kadb[n]: prompt (n is the number of the CPU that kadb is running on). At this point enter the command **cpu0$<cpu** and press return. You will see the cpu0 structure decoded, as shown below.

```
cpu0:
cpu0:        id          flags    thread     idle_t     lwp        callo
             0           1b       ff62d600   f0181ee0   f02af9d8   0
cpu0+0x18:   runrun      kprnrn   chsnlevel  dispthread
             0           0        -1         ff62d600
cpu0+0x20:   intr_stack  on_intr  intr_thread intr_actv
             f01a2fa0    0        f0199ee0   0
cpu0+0x30:   base_spl    caddr1   caddr2
             0           0        0
```

```
pkgtrans -s 'pwd'/spool /dev/diskette SUNWsst
```

Summary

This article has only scratched the surface with respect to packages and package handling. There is much more to know. The AnswerBook is the best source for more information on this topic. There is also the man pages, and of course, the white paper on the OPCOM FTP server.

A Real Time Watch Dog

Ron.Winacott@Canada.Sun.Com

Here is the problem: You are writing a real time (RT) program, and when you start it for the first time, your system hangs. But is it, or is the real time process in a tight loop, and nothing else can run? Real time scheduling priorities are higher than the system's kernel priorities, so it is possible to hang the system by making an error in your real time code.

This article shows you two ways to stop a runaway real time process. The first by running a watch dog program at a higher priority than the real time process you are testing, and the other by using kadb to bring the runaway process back into the time share (TS) class, where you can terminate it.

The Real Time Watch Dog Program

The real time watch dog program is called **rt_wd**. It is not extremely intelligent, having been written solely as an example for this article. It does, however, give you something you can use now, and can improve upon for your particular needs.

The program is started with an argument that sets the maximum seconds you want ANY real time process to accumulate on the cpu. The watchdog program must be run as root, so it can terminate any process in RT that uses more than its allowed time. The rt_wd program will place itself into the background, open /dev/console for output and start a new session. Once it's running, rt_wd will put itself into the RT scheduling class at the highest available priority, using the **priocntl(2)** system call. Using an itimer, it wakes up every 20 seconds and looks at /proc for RT processes.

When a RT process is found, the PIOCPSINFO ioctl (see man **proc(4)**) is used to get the total user+system cpu time. If that time is greater than half of the allowable limit, a warning is printed on the console. If the time is greater than the allowable limit, a SIGKILL is sent to the RT process.

Since the watchdog process runs at a higher priority than the testing RT process, the runaway process will be preempted when the itimer goes off, and the watchdog can terminate it. Even if the runaway process stays in user code (i.e.: never makes a system call), the signal will still be delivered as the preempted process leaves the dispatch queue.

Here is the source listing for the rt_wd program:

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/priocntl.h>
#include <sys/rtpriocntl.h>
#include <sys/signal.h>
#include <sys/fault.h>
#include <sys/syscall.h>
#include <sys/procfs.h>
#include <sys/time.h>
#include <sys/resource.h>

void sigtimeout(int sig);

int maxsec = 0;
int usedsec = 0;

int size;
struct rlimit rl;

struct itimerval it;
struct sigaction sigact;

main(int argc, char *argv[ ])
{
    pcinfo_t   pcinfo;
    pcparms_t  pcparms;
    pid_t      pid;
    int        i = 0;

    if (argc < 2) {
        fprintf(stderr,"usage: rt_watchdog Seconds\n");
        exit(1);
    }
    maxsec = atoi( argv[1] );
    if (maxsec == 0) {
        exit(1);
    }
    /*
    ** Put ourself into background, and into the RT
    ** scheduling class at the
    ** highest priority (59). with the priocntl syscall
    */
    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(1);
    }
    if (pid)
        exit(0); /* parent */
    rl.rlim_max = 0;
    getrlimit(RLIMIT_NOFILE, &rl);
    if ((size = rl.rlim_max) == 0)
        exit(1);
    for (i = 0; i < size; i++)
         (void) close(i);
    i = open("/dev/console", 2);
    (void) dup2(i, 1);
    (void) dup2(i, 2);
    setsid();
    /*
    ** put ourself into RT mode now
    */
    strcpy(pcinfo.pc_clname, "RT");
    if (priocntl(0, 0, PC_GETCID, (caddr_t)&pcinfo) == -1) {
```

device instance number for this system. The device is initialized on first attach and de-initialized (stopped) on last detach. Once in the DL_ATTACHED state, the user must send a DL_BIND_REQ to associate a particular SAP (Service Access Pointer) with the stream.

Applications should **not** hardcode to this particular implementation specific DLSAP address format but use information returned in the DL_INFO_ACK primitive to compose and decompose DLSAP addresses. The sap length, full DLSAP length, and sap/physical ordering are included within the DL_INFO_ACK.

In terms of the DLPI primitives, the driver supports: DL_UNITDATA_REQ, DL_ATTACH_REQ, DL_DETACH_REQ, DL_BIND_REQ, DL_UNBIND_REQ, DL_INFO_REQ, DL_PROMISCON_REQ, DL_PROMISCOFF_REQ, DL_ENABMULTI_REQ, DL_DISABLMULTI_REQ, DL_PHYS_ADDR_REQ, DL_SET_PHYS_ADDR_REQ.

The source for the psli driver is available on the OPCOM FTP servers.

Packaging for Solaris 2.x

Gerry.Singleton@Canada.Sun.Com

All software distribution on Solaris 2.x is by means of *packages* so that individual software products can be added or removed using standard utilities.

This article offers a simplified view on preparing a product for distribution. Readers who are interested in a more comprehensive description of the packaging process are invited to pick up an available white paper from the OPCOM FTP servers.

Structure of a Package

Packages require two mandatory files, in addition to the files that make up the product, to define their structure. These are the **prototype** file and the **pkginfo** file. The prototype file defines the package during the creation process while the pkginfo file provides the same service during installation. Thus the minimum package may consist of a pkginfo and the files of the software product. However, for illustration, the example used within this article uses some of the optional files in order to provide a more realistic picture of the packaging process. The files used for the example are **postinstall**, **sed**, **preremove** and **copyright**. Postinstall provides instructions to perform after installation or, if you prefer, to complete the installation. Preremove ensures that any previously installed packages of the same name are properly removed before new files are installed. The sed file provides sed instructions for modifying the specified files in the specified manner. Lastly, the copyright file which contains any copyright statements that relate to the package being installed.

There are many more optional files that may be used to describe packages. You should check your AnswerBook for detailed descriptions.

Preparation

In preparing a package, your files may be laid out in either of two ways; in a hierarchical structure or as a flat file system. This example assumes a flat directory structure . This is also the time to select a name for the package, such as SUNWsst, the name used with this example.

Making A Prototype File

Once the directory structure is in place, change to the directory where the package was created and use the following to create the prototype:

```
# cd <path>
# find . -print | pkgproto .=/usr/kernel/drv > prototype
```

The command sequence produces:

```
d none /usr/kernel/drv 0775 pms mts
f none /usr/kernel/drv/sst=sst 0664 pms mts
f none /usr/kernel/drv/sst.conf=sst.conf 0444 pms mts
f none /usr/kernel/drv/sst_def.h=sst_def.h 0444 pms mts
f none /usr/kernel/drv/sstest.c=sstest.c 0664 pms mts
```

Manually editing this file is the next step in creating the working prototype file. It is at this point that control files are added, permissions and ownership adjusted and paths corrected. The following is the prototype file that results:

```
i pkginfo
i postinstall
i preremove
i copyright
e sed /etc/devlink.tab ? ? ?
f none /usr/kernel/drv/sst=sst 0755 root sys
f none /usr/kernel/drv/sst.conf=sst.conf 0644 root sys
f none /usr/include/sys/scsi/targets/sst_def.h=sst_def.h 0644
bin bin
d none SUNWsst 0755 root sys f none SUNWsst/sstest.c=sstest.c
0664 root sys
```

Creating the Package for Distribution

Now that there is an accurate, complete prototype file, creating the package is the next step. First create a directory called spool. In the example, this would be /pkg/spool; i.e. where the control files are stored. Step two creates the package and deposits it in this local spool directory in preparation for distribution:

```
pkgmk -o -b 'pwd'/sst -d 'pwd'/spool -r 'pwd'
```

Next, using the **pkgtrans(1)** utility, the package is transferred to the distribution medium, which may be either tape or diskette, using:

New Products

Dave.Guerra@Canada.Sun.Com

On November 10th, 1992, Sun Microsystems Computer Corporation announced three new systems at both the low and high end of their product spectrum—the SPARCclassic™, SPARCstation™ LX and the SPARCcenter® 2000.

The SPARCclassic and SPARCstation LX are SMCC's new low cost color workstations. Packaged in an IPX™ like chassis, they are highly integrated for low cost, but with a 50Mhz MicroSPARC™ processor, they also provide advanced system performance. Some of the system highlights are:

- 50 Mhz MicroSPARC (Tsunami) processor for a 59 MIPS performance rating.
- 16-96 Mb of 64 bit main memory.
- 8 bit color frame buffer on the SPARCclassic and accelerated GXplus 8 bit color on the SPARCstation LX. The SPARCstation LX supports an optional 1 Mb of Video Ram for resolutions of up to 1600x1280. A new 15" SVGA monitor (1024x768) is also available.
- The SPARCclassic provides 8 bit audio, while the SPARCstation LX provides 16 bit audio, both from an internal speaker. An external speaker box is optional on the SPARCstation LX.
- Internal disk up to 1.05 Gb and up to 21 Gb of external disk on a 10 Mb/sec SCSI bus.
- 2 synchronous serial ports and 1 parallel port.
- Twisted pair ethernet with optional AUI thick ethernet.
- 2 Sbus slots are provided for peripheral expansion.
- SPARCstation LX supports ISDN (BRI) for digital connectivity.
- Only runs on the Solaris 2.1 operating environment.
- Tape Automated Bonding (TAB) is used in the manufacturing process to reduce system board real estate and increase system reliability. This allows SMCC to offer a 1 Year return-to-depot warranty on the system and attached peripherals.

The SPARCcenter 2000 series is SMCC's first multi-CPU board MP, designed for data center expandability, packaging, performance and system availability requirements. Its modular, balanced design utilizes standard components resulting in the most affordable, highest performance midrange server with industry-leading system throughput. Features include:

- A maximum of 10 CPU boards, yielding 2-20 way multiprocessing with SuperSPARC™ modules and 1 Mb of SuperCache™ per processor.
- Up to 5 Gb of main memory capacity with 512 Mb maximum per CPU board.
- Each CPU board contains 4 Sbus slots for maximum scalability and expansion. This allows a total of 40 Sbus slots when fully configured, and allows up to 1 terabyte of disk capacity.
- As CPU boards are added, so are memory and I/O, providing for a balanced growth and virtually eliminating any kind of bottlenecks.
- Uses a packet-switched XDBus™ developed by Sun and Xerox to provide a 640 MB/s peak and a 500 MB/s sustained performance at 40 Mhz.
- Advanced fault isolation detects any failure, shuts down and isolates the failure at the module or subsystem level — CPU, Memory, I/O, and system bus. ASR (Automatic System Reconfiguration) is then used to reboot with the new configuration.
- Packaging uses the same external cabinet as the 690MP system with many internal changes. Temperature, fan and voltage sensors monitor environment and shutdown the system before any damage occurs. The system cabinet supports up to 38 Gb of disk and three 8 mm tape drives.
- Expansion cabinets can be configured with either disk or tape providing for up to 100 Gb of disk per cabinet or 20-40 Gb of tape backup.
- Runs Solaris 2.1 and supports multi-threading and fully symmetric multiprocessing (SMP).

A Sample DLPI Compliant Driver

Srinath.Venkat@Corp.Sun.Com

This article describes a sample Pseudo Streams Multi-threaded DLPI driver(psli). The intended readers are those developers who need to write a multi-threaded DLPI driver. The developer can use psli and make only device and physical medium specific changes to create a new working DLPI driver.

It is assumed that the reader is familiar with SunOS 5.X Multi-thread Architecture, the SunOS 5.x Device Driver Interface/ Driver Kernel Interface (DDI/DKI), STREAMS and the Data Link Provider Interface (version 2).

DLPI enables a data link service user to access and use any of a variety of conforming data link service providers without special knowledge of the provider's protocol. The DLPI primitives are defined in terms of STREAMS messages. All DLPI messages are either M_PROTO or M_PCPROTO type and contain the appropriate DLPI message structure in the leading mblk.

DLPI supports the concept of a DLSAP address which consists of a physical address(ppa) plus a datalink service access point (SAP) address. The driver writer must specify the DLSAP address format accepted by the driver. The psli driver is a "style 2" Data Link Service provider. An explicit DL_ATTACH_REQ message by the user is required to associate the opened stream with a particular device (ppa). The ppa ID is interpreted as an unsigned long and indicates the corresponding device instance (unit) number. An error (DL_ERROR_ACK) is returned by the driver if the ppa field value does not correspond to a valid

The architecture of the program is that it is designed to read data from an input file, then write the data to a number of output files, as specified on the command line. Finally a loop was inserted to run an arbitrary calculation to use the idle CPU time while the I/O was being performed. Of course the demo had to run both multi-threaded and single-threaded so the two cases could be compared. Once developed, the program was run in a variety of situations to measure how much performance gain could be achieved by threading, and in what type of situations it was best suited.

Of course, the results varied based on the situation that was specified. Using a large loop count on the calculation would cause the calculation to take so long that it dragged on long after the I/O had completed and showed very little benefit. The CPU was still the bottleneck. Similarly, little or no calculation would still leave the CPU idle waiting for the I/O to complete. However, the benefits in more balanced cases were beyond expectations. Many of the multi-threaded tests ran in half the time of the equivalent single-threaded run and the best cases ran in ONE THIRD the time! As you can imagine, this was good news to our Reseller, who was worried that by multi-threading their application to take advantage of multi-processor hardware, they might penalize their users who ran on uni-processor machines.

The demo program also helped us learn which situations would be most profitably threaded, and how to best do it. For example, when we adjusted the amount of calculation that was being done, the highest results came when the system and user time reported by /bin/time were roughly equal. The most dramatic results came when the wait for I/O completion was the longest. For example, accessing files on an NFS server, or waiting for a slow RPC server (as in the Corporate oil prices demo), or forcing data to be flushed to disk using the fsync call are the type of activities that generate dramatic results.

The demo also showed that the best performance was obtained when the number of LWPs was roughly equal to the number of tasks being performed at once. If we were reading one input file and writing to four output files, plus one calculation loop, the best performance came when the number of LWPs was close to 6 (this was controlled using the **thr_setconcurrency(3)** routine.) Use of 1 LWP would usually behave identically to the single-threaded case since there was only one schedulable entity for the kernel to work with. Although the threads library has some ability to adjust the number of LWPs available, it is often possible to control this parameter manually to get better performance.

Use mttool as a Selling Tool

As an outgrowth of this project, the original demo program has been given an OLIT front-end and packaged for more general use. It has a screen showing the progress towards completion for each of the tasks created by the demo. This can very effectively demonstrate how the multi-threading works with all tasks being processed in parallel with multi-threading turned on, while they are performed sequentially with single-threading. Another screen graphs the time the tests run when single-threaded versus when multi-threaded so you can easily see the speedup that threading provides.

Like the original demo program, the demo-tool has a lot of parameters that may be dynamically configured so that an attempt may be made to emulate some target application by trying to match the I/O and calculation characteristics. In this way, it is possible to get a rough idea of how much speedup that application may be able to achieve through threading. This can be useful when talking to specific customers to demonstrate that even their application can benefit from threading.

The demo-tool is available to those people on SWAN by anonymous ftp of the file /tmp/mtdemo.tar.Z from gesundheit.West. It is also available on the OPCOM FTP servers, in /pub/demos/mtdemo.tar.Z

Conclusions

Sun has charted a course based on the idea that the number of MIPS shall double once a year. Multi-processors are a central part of Sun's strategy for achieving this goal. With this technology, Sun should be able to continue producing exponentially faster hardware through this decade and beyond. But for customers to be able to fully realize the benefits of the faster hardware, their applications will need to be threaded.

Many people might be tempted to wait to do multi-threading until SPARCstation 10 model 52s and 54s become commonplace because of the traditional connection between multi-processing and multi-threading. Our work, meanwhile, has concentrated on the benefits of multi-threading to uni-processor platforms. We have seen good results on this unexpected front. A whole new class of applications shows significant speedup due to multi-threading and these applications are in a position to realize these advantages as soon as they can integrate in the user level threads library. So we think there is enough benefit to push multi-threading now to get the momentum building even at this early stage.

In the long run, having more and more of the SPARC applications multi-threaded will be a tremendous competitive advantage for Sun. We have a head start with multi-processing on the desktop and it will cause these applications to run faster on Sun than other platforms. This will help Sun when real-life benchmarks are run using these applications against our competitors and it will help the Reseller against their competition.

to read data from an input file. None of these tasks actually cause the single CPU to do more - for that you need multi-processing - but they can use the single CPU more efficiently by taking advantage of what would otherwise be idle time.

First Impressions

This is the feature of threads that first interested us. We have been working with one of Sun's Resellers, trying to position them to be able to take advantage of the user-level threads technology as soon as it is available to customers— sometime in Spring, 1993. It was clear that the compute-bound functions of this application would clearly benefit from a having a threaded implementation on a multi-processor workstation. However, we believed there was potential benefit from threading on a uni-processor system as well. Due to the large size of the data used by their application, occasionally reaching into the gigabytes, it cannot just store all the data into memory and so is very I/O-intensive, in addition to being computatively intensive. Finally, the application is an interactive program that must continually respond to the user and update the information being displayed. It appeared that there was considerable opportunity to overlap these three classes of processing with each other.

The Thread API and "Porting"

We knew that our Reseller was worried about a "porting" effort. The simplicity of the threads API was very attractive. In simple cases, threads may be used effectively knowing only one function: **thr_create(3)**. This function creates a new thread and invokes the indicated function to execute within that thread. To exit the thread, the function need merely return. The following code

```
thread_t thread_id;
if (thr_create( NULL, NULL, mysub, myarg, 0, &thread_id ) < 0 ) {
    perror( "thr_create" );
}
```

would invoke the routine mysub running in its own thread:

```
void* mysub( void* myarg ) {
    .
    .
    .
    return;
}
```

The threads library has been designed to *do the right thing*. For example, the library can adjust the number of LWPs available, in addition to caching LWPs for quick reuse. While there are entry points that allow the programmer to manually adjust the number of LWPs and control priorities and other such fine-tuning, it is often not needed.

Simplicity in Programming

The effect is that, in general, the straightforward, simple approach is usually the best. At the most basic level, using threads became as simple as substituting a call to thr_create(3) for a call to an existing routine. The following code fragment uses an #ifdef to contrast the "before" and "after".

```
#ifdef USE_THREADS

thread_t thread_id;

if( thr_create( NULL, NULL, mysub, myarg, 0, &thread_id ) < 0
    perror( "thr_create" );
}

#else

mysub( myarg );

#endif
```

Synchronization Facilities

Trying to develop a program that could potentially have several threads all running simultaneously is very similar to developing signal handlers. In particular, you have to worry about multiple threads trying to change data at the same time, and other similar problems. The threads library, in addition to the basic thread API, also provides several types of synchronization facilities to handle these situations. As their name suggests, these facilities are mechanisms to allows different threads to synchronize, or cooperate, with each other.

There are four basic synchronization types offered as part of the threads library. A **mutex** is a lock that may be held by only one thread at a time, and is very useful in enforcing that your thread is the only one changing a piece of global data at any given time. **Semaphores** are used as counters, and in particular, can be used to keep track of the availability of a finite set of resources such as I/O buffers when there are multiple threads all doing I/O at one time. **Condition variables** offer an extremely powerful mechanism to wait for some arbitrary condition that is defined by the programmer. Finally, **readers/writer locks** provide the ability to control changing of data like a mutex does, but can additionally allow multiple threads to access the data read-only. For an in-depth explanation of these synchronization facilities, see the whitepaper on SunOS Multithread Architecture, available on the OPCOM FTP Server.

Demonstrating the Benefits - mttool

Armed with an early access version of the threads library, and not a whole lot more information about threads than what has been described above, we were able to develop a demo program to show the value of threads on a uni-processor. The idea of the demonstration was to show the benefits of overlapped I/O. At the time of the demo, we only had access to Solaris 2.0, which limited us to a uni-processor Sun, but this matched very well with the predominant type of machine used by our Reseller's customer base.

7) After the installation is complete, use neworld to restore your original system configuration. As with oldworld, first run it without the -y flag to verify that it is doing everything that you want it to do. After running oldworld, the system should be rebooted in order for the new configuration to be effective.

8) Restore all software saved in Step 4.

9) Run moveme and scriptran for all users and root to convert shell scripts, start-up scripts and user configuration files.

10) If any problems occur due to the system reconfiguration carried out by neworld, moveme or scriptran, it is possible to revert those items. The utilities back up all files before they modify them. The backup files are located in the same directory with the extension .sabak. Changes can be undone by copying back the original file or editing the new file. Once you are satisfied with the system configuration, all backup files can be removed with the command:

```
find / -fstype nfs -prune -o -name *.sabak -print | xargs rm
```

Thanks and kudos to Mr. David Taber of SMCC and his team who put this Toolset together. In particular, the excellent documentation without which this article could not appear. The Admigration Toolset currently supports only Sun workstations with the sun4c architecture. Support for other architectures is planned to coincide with the release of Solaris 2.1. The Admigration Toolset may be found on the OPCOM FTP servers.

MTTOOL White Paper

John.Richardson@West.Sun.Com and
Edward.Milstein@West.Sun.Com

"To Thread or not to Thread? That is the Question our End-users, Resellers, and ISVs are Asking!"

Introduction

Threads - what are they? What do they do? What applications and hardware platforms benefit most from threads?

Can threads with Solaris 2.x help me win, be more competitive? Can threads be my company's Competitive Advantage?

This is a short list of the typical questions our customer base, software developers and resellers have been asking. This article was written to help us all understand where and when threads should be used. Threads are our competitive weapon! It is up to all of us to recognize the advantage threads provide.

"We must take advantage of the advantage (threads) to WIN!"

The *mttool* Demonstration Program

Here we will present and discuss a locally developed demonstration tool that allows an interested party to use and understand threads and the advantage that threads provides. The demo tool is completely configurable (through a configuration panel), to allow a user freedom in the degree and granularity of the execution.

Background

Already there has been a lot of interest in user level threads. There was a good article by Venkat Srinath describing the basic architecture of Sun's user level threads library. For details on the architecture we would recommend reading this article. It appeared in the second SunOpsis, dated September, 1992.

To date, many copies of the early access version of the User Level threads library have been distributed. I expect most of us will immediately appreciate the performance gains possible with multi-processing hardware. We need to understand and position our customers to be able to take advantage of this technology as soon as it becomes available to the end user.

Threads are Elegant

Solutions to the problem of a program utilizing multiple processors to achieve greater application throughput is effective on uni-processor hardware as well as multi-processor hardware. This is often for different reasons. With a uni-processor platform one might want to break a program into different tasks so that one task is able to run while another task is blocking on I/O. This prevents the process sitting idle, thereby achieving great host efficiencies. Many of the past attempts to do this have created multiple processes and led to the development of all sorts of complicated mechanisms for interprocess communication such as shared memory, message queues, and semaphores. In fact, the paradigm offered by the threads library is often more natural than many of these "kludges" developed for dealing with the sharing of data between different tasks within a single program. Therefore programmers are attracted to the threaded paradigm for reasons beyond mere performance.

Overlapped IO and Interactive Computing

An example of this situation might be an interactive program which could be made more responsive if it could have one task to monitor the user interface even while another task is waiting for disk I/O to complete. In addition, a network server program might like to have one task for each client, so that when the current client is waiting, the server is able to respond to another, offering faster response time. A compute intensive program might do calculations in one task, while another task is waiting

File conversions using moveme are accomplished in two ways. The first method utilizes translation tables. For the selected file, a translation table provides a string substitution facility through a list of source patterns which are replaced by result patterns. The second method of file conversion requires moveme to call a translation program, providing maximum flexibility. The translation program method of file conversion is used to convert shell scripts. In these instances, moveme is instructed by the control file to execute scriptran.

Scriptran

Scriptran is the utility used by moveme to convert shell scripts so that they are Solaris 2.x compliant. Since moveme only runs scriptran on the standard user files (.cshrc, .login, etc.), you must run scriptran on any remaining shell files.

Scriptran converts any shell script, whether it is csh, sh, ksh or tcsh; it converts the scripts by looking for lines where system commands are called. The calls are checked for the following conditions:

- the command is now obsolete.
- the command is in a new location and the new location is not in the user's path.
- the command is using a flag that is now obsolete.
- the command is using a flag which has changed meaning or has become a different flag.
- output of the command or flag has changed, and the output is being redirected.

Where differences occur, scriptran attempts to fix the script so that it becomes compliant. Should a fix be impossible, a warning message is generated along with instructions to fix the problem manually.

Whatnow

Whatnow provides a brief synopsis of the differences between Solaris 1.x and Solaris 2.x approaches to the specified command. Information is also supplied on devices, automounter, printing, export, vfstab options, and BCP. To obtain a whatnow summary on a given topic, type "whatnow *topic*".

The System Conversion Roadmap

The following is a summary of the steps you need to take with the Admigration Toolset in order to migrate from Solaris 1.x to Solaris 2.x. You will discover a wealth of detail in the documentation that accompanies the package.

1) Decide which systems to upgrade to Solaris 2.x, and which to maintain at Solaris 1.x. The following factors are relevant to this decision:

   - If there are third-party applications that are not ported to Solaris 2.x, maintain some systems in Solaris 1.x to run these applications. This includes application code that uses BSD kernel interfaces or system files.
   - Consider devices that require special drivers, or libraries that are not yet available on Solaris 2.x.
   - If you still need to run Sunview, you will need Solaris 1.x. The Sunview environment is not available in Solaris 2.x.
   - It is advisable to leave some systems with Solaris 1.x until all users become comfortable with Solaris 2.x.
   - Note if the system is a Sun-3 or is acting as a server for a Sun-3. Sun-3's are not supported under Solaris 2.x.

2) Determine the configuration with which to install Solaris 2.x, and the system parameters to convert using oldworld and neworld. Consider the following issues:

   - Whether to use oldworld/neworld to automatically convert diskless clients, or to recreate them when installing Solaris 2.x. If a diskless client is not of the same architecture, or is not being upgraded to Solaris 2.x, do not use oldworld/neworld.
   - Which software to install; and, whether or not to include the Berkeley Compatibility Package (BCP).
   - Whether or not to upgrade to NIS or NIS+.
   - Whether the same disk layout should be used or restructured.
   - Whether or not to port the printer configuration. If you want to reconfigure the printers to an SVR4 structure instead of a BSD structure, you may want to configure them manually.

3) Run oldworld on the system to save the system configuration parameters that you need. If you choose not to convert all of the parameters, use the -i flag to selectively choose the parameters. Run oldworld first without the -y flag to verify that it is porting everything that you need.

4) This is the time to save all remaining user files and third-party software (at least those that can be used with Solaris 2.x). This step can normally be done with the tar, cpio or dump[1] commands. The output log of oldworld includes a list of files and directories that you should save. **Do not back up any system files in the root or /usr directory**. Restoring these would not only conflict with oldworld/neworld and cause erroneous results but would likely force you to re-install Solaris 2.x.

5) Refer to the Solaris 2.x Installation and Configuration manual for instructions on installing the new operating environment.

6) At this point you are ready to install Solaris 2.x. Check to make sure that you are ready to initiate the install; after the process has begun, it is impossible to turn back without a full restore or reinstall.

1. dump is changed to ufsdump on Solaris 2.x

Admigration Toolset, the task is greatly simplified and eases the migration to Solaris 2.x.

There are five utilities in the Admigration Toolset, each of which is designed to simplify the migration of a Sun workstation from Solaris 1.x to Solaris 2.x. First, the **oldworld** utility saves the system configuration and customizations used under Solaris 1.x. Once Solaris 2.x is installed the **neworld** utility automatically restores and converts this saved configuration. **Moveme** and **scriptran** handle conversion of user files and scripts to meet the requirements of Solaris 2.x, respectively. Lastly, the **whatnow** utility is a reference utility that provides on-line translations between 1.x syntax and 2.x syntax.

Once your system is reconfigured using the Admigration Toolset and Solaris 2.x is running, you will have the opportunity to familiarize yourself with the SVR4 configuration by comparing it to the previous configuration. You can also compare the two configurations on a case-by-case basis.

Using the Toolset

The oldworld/neworld utilities are used to port your system configuration across the upgrade from Solaris 1.x to 2.x. Before Solaris 2.x is installed, the system administrator must run oldworld on the system. Oldworld saves all files necessary for the system configuration to an intermediate device (the default is floppy disk). After Solaris 2.x is installed, neworld is executed to reconfigure the system. Neworld reads the files created by oldworld and converts and reconfigures them to the new SVR4 formats. When finished, the Solaris 2.x system is configured as closely as possible to the Solaris 1.x configuration. However, as with any change, it's still important to make proper backups.

The system configuration items that are saved for conversion include:

- System configuration files, such as kernel configurations
- Users and groups
- Database files for aliases, automount, disk format, terminals and printers
- System start-up scripts
- Remote mounts and exports
- Diskless clients
- User files found in system directories

The following items are not saved by oldworld:

- Non-system directories and user directories
- Devices, device drivers and kernels (not the same)
- Large files in system directories (saves space on the floppy)
- System and user files on remotely mounted directories
- NIS maps that are on remote systems
- Diskless clients that do not have the same architecture, or that are not being upgraded to Solaris 2.x
- Local filesystem mounts

Running Oldworld

You should run oldworld in two passes. On the first pass, run it without the -i and -y flags. Doing so creates and/or prints a log detailing which items would be saved without actually performing the save. After you review the log to determine whether you are including all the items you want to save, run oldworld with the -y flag. If you wish more interaction or you do not want to save everything, include the -i flag and oldworld will prompt for confirmation before saving each item.

If, at any time, you are uncertain about a specific item, save it anyway; you may cancel the conversion of that specific item during the neworld operation.

Running Neworld

Run neworld after the Solaris 2.x install is complete. As with oldworld, you should run neworld in two passes. On the first pass, run it without the -y flag. Review the log that is generated to verify that it is converting everything you require. At the same time, check for error and warning messages. For each item that produces an error or warning message, you must decide whether to change the specific system configuration manually, or to fix the problem after neworld upgrades it. After making the appropriate decisions, run neworld with the -y flag and, perhaps, the -i flag.

Neworld saves all system configuration files that it modifies in an in-place backup. If there is any problem with one of the files, just copy the original file back.

Moveme

Moveme converts configuration files found in user home directories (e.g., login, .cshrc, etc.) to formats compatible with Solaris 2.x. These format conversions are controlled by the *moveme control file*. You may modify the moveme control file to incorporate any desired file conversions, including those not currently supported. Moveme must additionally be supplied with a list of users whose home directories are to be processed. This allows for batch mode conversions.

Moveme currently processes the following files by default: .cshrc, .login, .profile, .openwin-init, and .xinitrc.

If you are converting users, in addition to yourself, you must have root privileges on the system where these users have their files. Under normal circumstances the system administrator (logged in as root on a server which hosts user home directories) will run moveme. Of course, individual users may also run moveme on their own files.

SUN MICROSYSTEMS COMPUTER CORPORATION

SunOpsis

The Solaris® 2.x Migration Support Center Newsletter

1-800-363-6200

VOLUME 1, NUMBER 4, November 1992

Editorial

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

As many of you are aware, November has been a busy month with respect to new product announcements. All of the new machines can only be run with Solaris 2.1. Please see the New Products article in this issue for more details.

Due to some internal changes here at Sun, you may see a slight disruption of our Internet Ftp server. This service is moving to a commercial facility which may cause some down time for a day or two. We expect the service to be fully operational at the beginning of December. The hostname will remain the same (opcom.sun.ca). During this time, you can still access the anonymous FTP archives via email. Please see the OPCOM FTP Server in this issue for email details. Note that the Sun internal server is not affected by this move.

With the holiday season upon us, I thought it would be appropriate to let you know our hours of operation in December:

Dec 24, 1992	closing at 5:00pm Eastern Time
Dec 25, 1992	closed all day
Jan 1, 1993	closed all day

Standard hours apply at all other times.

As always we appreciate your comments or suggestions.

The Admigration Toolset

Gerry.Singleton@Canada.Sun.Com

Introduction

Migrating from one release of an operating system to another can seem rather difficult. When the new release differs so greatly from its predecessor, the process is made even more difficult, sometimes overwhelming. The September issue of SunOpsis included an article that attempted to address this problem in a rational manner. This article will introduce the reader to an easier solution: the Admigration Toolset from SMCC. Simply put, the Admigration Toolset simplifies the task of upgrading Sun workstations to Solaris 2.x.

As described in the September article, system administrators normally have to know both SVR4 and BSD system administration in order to configure their systems. Now, with the introduction of the Admigration Toolset, this is no longer necessary. By using the