sun microsystems

Software Technical Bulletin
April 1987

Software Information Services

Part Number 812-8701-03
Issue 1987-3
2 April 1987

sun
microsystems

Software Technical Bulletin
April 1987

Software Information Services

Part Number 812-8701-03
Issue 1987-3
2 April 1987

Software Technical Bulletins are distributed to customers with software/hardware or software only support contracts. Send comments or corrections to ''Software Technical Bulletins'' at Sun Microsystems, Inc., 2550 Garcia Ave., M/S 2-34, Mountain View, CA 94043 or by electronic mail to sun!stb-editor. Customers who have technical questions about topics in the Bulletin should call Sun Customer Software Services AnswerLine at 800 USA-4-SUN.

Contents

NOTES & COMMENTS

1

NOTES & COMMENTS

Managing Editor's Notes

As promised in our last issue, this month we start a new feature, a question and answer column. We hope you will send your ideas and questions for this feature to *sun!stb-editor*. We cannot promise to answer each and every one, but we do promise to read them all and choose some 'juicy questions' to be answered.

Another item from last month was the expansion of the Software Information Services (SIS) group. I am happy to report that since last month, we have hired our first senior technical writer, and, even more good news, our second senior technical writer will be starting April 13, 1987. Between them, they bring over 15 years technical and writing experience to the Software Technical Bulletin (STB).

We are all very excited about what the future holds in store for the STB. What the future holds in store today, in fact, is next year's budget proposal.

At the last group meeting we sat down and tried to picture what it would be like publishing the January 1989 issue of the STB. We envisioned color, great graphics, and expanded issues focusing on Sun's growing product line. We even wondered if there could be STB personals. Something like, 'Lonely Graphics Processor seeks non-smoking Graphics Buffer.'

Our goal remains to provide timely technical software information to our support contract customers. So please keep your comments and ideas coming.

Thanks.

The Managing Editor

2

CREATION OF THE U.S. ANSWER CENTER

2

CREATION OF THE U.S. ANSWER CENTER

Phone Support Reorganization

Phone Support Reorganization

The Customer Software Services organization has just announced a reorganization of the phone support function that will better serve our support contract customer base.

Product Groupings

The new structure features groups that are organized around product categories. These are:

- UNIX
- Graphics
- Languages
- Data Communications
- Applications (e.g. Sun Ingres, Alis)
- Personal Answer Line

U.S. Answer Center

Collectively, these groups will now report to the newly created position of U.S. Answer Center manager. The USAC manager will report directly to the Director of Customer Software Services.

Why the Change

This change is being made to allow more technical depth in specific areas rather than the generalist approach that was in place before. It will also allow much room for expansion, flexibility and growth as customer requirements change.

This organizational change will have no impact on our current contractual commitments and fits in with our already announced services (AnswerLine and Personal AnswerLine). Nor does it change the way you contact Sun for your service.

Customer Software Services expects this change to improve our productivity and, we hope, improve customer satisfaction. We are trying to make this change as smoothly as possible, but should you find any temporary inconvenience, please feel free to send us your comments, and remember that we are convinced this change will bring about a permanent improvement.

3

ARTICLES

3 ARTICLES

Sockets

Socket Programming Explanations and Examples

This article hopefully will clarify using `SIGIO` and `SIGURG` with sockets and provides a couple of socket programming examples. Sun Software Information Services has been getting a lot of customer calls on this topic. Therefore, we will provide ongoing STB articles relating to sockets, dealing with one or two topics at a time. We will cover two topics in this article -- Sockets: Asynchronous I/O and Out-of-Band Data, and Asynchronous I/O and Internet Domain Stream Sockets.

Sockets: Asynchronous I/O and Out-of-Band Data

The term *asynchronous I/O* used in connection with a socket refers to notifying the process or the process group associated with the socket when I/O is ready. This is done asynchronously and via a `SIGIO` signal. This facility is useful when you do not want to poll for pending I/O. An example is when the process wants to perform other functions while waiting for I/O ready.

There are several ways to request asynchronous notification of I/O on a socket descriptor by using `ioctl(2)` or `fcntl(2)` or both.

First, either the process or the process group associated with the socket must be set to receive the `SIGIO` signal. Use one of the system calls shown below to set the process group to receive the `SIGIO` signal. Note that the process group can be set to receive `SIGURG` signals using the same system calls.

```
int pid;

pid = -getpid();
if (fcntl(sock, F_SETOWN, pid) < 0)
        perror("fcntl: F_SETOWN");
if (ioctl(sock, FIOSETOWN, (char *)&pid) < 0)
        perror("ioctl: FIOSETOWN");
if (ioctl(sock, SIOCSPGRP, (char *)&pid) < 0)
        perror("ioctl: SIOCSPGRP");
```

Second, you need to set up the socket to receive asynchronous notification by using one of the system calls shown below.

```
        int val;

        if (fcntl(sock, F_SETFL, FASYNC) < 0)
                perror("fcntl: F_SETFL FASYNC");
        val = 1;
        if (ioctl(sock, FIOASYNC, (char *)&val) < 0)
                perror("ioctl: FIOASYNC");
```

Example Server and Client Programs

The following server and client programs illustrate using asynchronous notification of I/O under the Internet domain stream socket abstraction. The server first creates an Internet domain stream socket. The system selects TCP as an appropriate protocol. The server binds a name to the created socket, lets the system select a port number, and then obtains the port number.

The server listens in its main loop for a client to connect to the named socket. A new socket is created upon accepting a connection. The server then writes a message to the client on the new socket at two-second intervals. When the server receives a SIGPIPE signal it knows that the client has disconnected. This signal results from writing on the socket without a client to read it. The server then begins listening again for another client connection. This procedure is then repeated.

The server is terminated by a SIGQUIT signal, at which time the full-duplex connections on the two sockets are shut down and the socket descriptors are closed.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/signal.h>
#include <netinet/in.h>

int Sig = 0;

char Buf[] = "Eat some chocolate!0;
int Sock = 0;
int Msgsock = 0;

/* The Server */
main()
{
        int length;
        struct sockaddr_in server;
        int sigpipe(), sigquit();

                /* Create Internet domain stream socket,
	 * letting the system select an appropriate protocol
	 */

if ((Sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
	perror("creating Internet domain stream socket");
	exit(2);
}

	/* Assign name to socket, letting system pick port */

server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = 0;
if (bind(Sock, &server, sizeof(server))) {
	perror("binding Internet stream socket");
	close(Sock);
	exit(2);
}

	/* Get assigned port number */

length = sizeof(server);
if (getsockname(Sock, &server, &length)) {
	perror("getting socket name");
	close(Sock);
	exit(2);
}
printf("Socket has port #%d0, ntohs(server.sin_port));

	/* Handle writing to a non-existent client */

signal(SIGPIPE, sigpipe);

	/* Terminate server on a SIGQUIT signal */

signal(SIGQUIT, sigquit);

	/* Continue to listen for connections */

for (;;) {
	if (listen(Sock, 5)) {
		perror("listen");
		close(Sock);
		exit(2);
	}
	if ((Msgsock = accept(Sock, 0, 0)) < 0) {
		perror("accept");
		close(Sock);
		exit(2);
```

```
                }
                do {
                        if (write(Msgsock, Buf, strlen(Buf)) != strlen(Buf))
                                perror("writing stream message");
                        sleep(2);
                } while (!Sig);
                close(Msgsock);
                Msgsock = 0;
                Sig = 0;
        }
}

/* We'll receive a SIGPIPE signal if the client is killed.
 * Handle it so we can accept future connections from new clients.
 */
sigpipe (s, code, scp)
int s, code;
struct sigcontext *scp;
{
        printf("received SIGPIPE0);
        Sig++;
}

sigquit (s, code, scp)
int s, code;
struct sigcontext *scp;
{
        if (shutdown(Sock, 2))
                perror("shutdown Sock");
        close(Sock);
        if (Msgsock) {
                if (shutdown(Msgsock, 2))
                        perror("shutdown Msgsock");
                close(Msgsock);
        }
        exit(0);
}
```

Asynchronous I/O and Internet Domain Stream Sockets

The client creates an Internet domain stream socket and the system selects TCP as an appropriate protocol. The client then initiates a connection by supplying the destination host name and port number obtained by the server. The client then sets up the process group associated with the socket to receive asynchronous notification of input on the socket. Upon receipt of a SIGIO signal, the client prints the server message. Finally, the client is terminated by a SIGQUIT signal, at which time the full-duplex connection on the socket is shut down and the socket descriptor closed.

More Example Programs

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <netdb.h>

#define BUFSIZE 512

int Sock;
int Sigio = 0;
int nSigio = 0;

/* The Client */
main (argc, argv)
int argc;
char **argv;
{
        struct sockaddr_in server;
        struct hostent *hp, *gethostbyname();
        int sigio(), sigquit();
        int n, val, pid;
        char buf[BUFSIZE];

        if (argc != 3)
        {
                printf("Usage: %s hostname portnumber0, argv[0]);
                exit(2);
        }
                /* Create Internet domain stream socket,
                 * letting the system select an appropriate protocol
                 */

        Sock = socket(AF_INET, SOCK_STREAM, 0);
        if (Sock < 0) {
                perror("creating Internet domain stream socket");
                exit(2);
        }

                /* Connect socket using host, port specified on command line. */

        server.sin_family = AF_INET;
        hp = gethostbyname(argv[1]);
        bcopy(hp->h_addr, &(server.sin_addr.s_addr), hp->h_length);
        server.sin_port = htons(atoi(argv[2]));

        if (connect(Sock, &server, sizeof(server)) < 0 ) {
                close(Sock);
                perror("connecting stream socket");
                exit(2);
        };

                /* Set process group to receive SIGIO signals
                 * when there is input on socket Sock
                 */

        val = 1;
        if (ioctl(Sock, FIOASYNC, (char *)&val) < 0)
                perror("ioctl: FIOASYNC");

        pid = -getpid();
        if (ioctl(Sock, SIOCSPGRP, (char *)&pid) < 0)
                perror("ioctl: SIOCSPGRP");

        signal(SIGIO, sigio);
        signal(SIGQUIT, sigquit);

        for (;;)
        {
                /* Wait for a SIGIO signal indicating input pending */
                /* (Actually client would be doing some useful work here
                 *  rather than waiting)
                 */
                sigpause(SIGIO);
                if (Sigio) {
                        if ((n = recv(Sock, buf, BUFSIZE, 0)) > 0) {
                                buf[n] = '  ';
                                printf("%s", buf);
                        }
                        Sigio = 0;
                }
        }
}

sigio ()
{
        Sigio++;
        nSigio++;
}

sigquit ()
{
        printf("Quitting, %d SIGIOs0, nSigio);
        if (shutdown(Sock, 2))
                perror("shutdown");
        close(Sock);
        exit(0);
}
```

An easy way to run this example is to execute the server and client in two separate windows. Start up the server first in one window and the server in another window. Supply the host name and port number printed by the server as arguments to the client. To stop the client, type `control-\`, or kill it with `kill -3`.

Under the Internet domain stream socket abstraction, a process group can be set to send or receive out-of-band data on the same pair of connected stream socket descriptors as normal data. Out-of-band data transmission is limited under Release 3.2 to one character. Both normal and out-of-band data are received via the same descriptor if the process is set up to receive asynchronous notification of I/O on a socket descriptor. `SIGIO` signals will be received on the OOB data as well as `SIGURG` signals.

In the following examples, the server sends the client both normal and out-of-band data. These programs can be executed in the same manner as the `SIGIO` example above.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/signal.h>
#include <netinet/in.h>

int Sig = 0;

char Buf[] = "Eat some chocolate!0;
int Sock = 0;
int Msgsock = 0;

/* The Server */
main()
{
        int length;
        char oobmsg;
        struct sockaddr_in server;
        int sigpipe(), sigquit();

                /* Create Internet domain stream socket,
                 * letting the system select an appropriate protocol
                 */

        if ((Sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                perror("creating Internet domain stream socket");
                exit(2);
        }
        /* Assign name to socket, letting system pick port */

server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = 0;
if (bind(Sock, &server, sizeof(server))) {
        perror("binding Internet stream socket");
        close(Sock);
        exit(2);
}

        /* Get assigned port number */

length = sizeof(server);
if (getsockname(Sock, &server, &length)) {
        perror("getting socket name");
        close(Sock);
        exit(2);
}
printf("Socket has port #%d0, ntohs(server.sin_port));

        /* Handle writing to a non-existent client */

signal(SIGPIPE, sigpipe);

        /* Terminate server on a SIGQUIT signal */

signal(SIGQUIT, sigquit);

        /* Continue to listen for connections */

for (;;) {
        int i = 0;

        if (listen(Sock, 5)) {
                perror("listen");
                close(Sock);
                exit(2);
        }
        if ((Msgsock = accept(Sock, 0, 0)) < 0) {
                perror("accept");
                close(Sock);
                exit(2);
        }
        do {
                        /* write normal data to client */

                if (write(Msgsock, Buf, strlen(Buf)) != strlen(Buf))
                        perror("writing stream message");
                        sleep(3);

                                /* send Out-of-Band data to client */

                        oobmsg = ('a' + i) &0177;
                        printf("sending OOBMSG %c0, oobmsg);
                        if ((send(Msgsock, &oobmsg, 1, MSG_OOB)) < 0)
                                perror("sending OOB data");
                        i++;
                        sleep(3);
                } while (!Sig);
                close(Msgsock);
                Msgsock = 0;
                Sig = 0;
        }
}

/* We'll receive a SIGPIPE signal if the client is killed.
 * Handle it so we can accept future connections from new clients.
 */
sigpipe (s, code, scp)
int s, code;
struct sigcontext *scp;
{
        printf("received SIGPIPE0);
        Sig++;
}

sigquit (s, code, scp)
int s, code;
struct sigcontext *scp;
{
        if (shutdown(Sock, 2))
                perror("shutdown Sock");
        close(Sock);
        if (Msgsock) {
                if (shutdown(Msgsock, 2))
                        perror("shutdown Msgsock");
                close(Msgsock);
        }
        exit(0);
}

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <fcntl.h>
```

```
#include <netdb.h>

#define BUFSIZE 512

int Sock;
int Sigurg = 0;
int nSigurg = 0;

/* The Client */
main (argc, argv)
int argc;
char **argv;
{
        struct sockaddr_in server;
        struct hostent *hp, *gethostbyname();
        int oobdata(), sigquit();
        int n, pid;
        char buf[BUFSIZE];

        if (argc != 3)
        {
                printf("Usage: %s hostname portnumber0, argv[0]);
                exit(2);
        }
                /* Create Internet domain stream socket,
                 * letting the system select an appropriate protocol
                 */

        Sock = socket(AF_INET, SOCK_STREAM, 0);
        if (Sock < 0) {
                perror("creating Internet domain stream socket");
                exit(2);
        }

                /* Connect socket using host, port specified on command line. */

        server.sin_family = AF_INET;
        hp = gethostbyname(argv[1]);
        bcopy(hp->h_addr, &(server.sin_addr.s_addr), hp->h_length);
        server.sin_port = htons(atoi(argv[2]));

        if (connect(Sock, &server, sizeof(server)) < 0 ) {
                close(Sock);
                perror("connecting stream socket");
                exit(2);
        };

                /* Set process group to receive SIGURG signals
                 * when there is Out-of-Band data on socket Sock
                */

        pid = -getpid();
        if (ioctl(Sock, SIOCSPGRP, (char *)&pid) < 0)
                perror("ioctl: SIOCSPGRP");

        signal(SIGURG, oobdata);
        signal(SIGQUIT, sigquit);

        for (;;)
        {
                char mark;

                /* read normal data */
                if ((n = recv(Sock, buf, BUFSIZE, 0)) > 0) {
                        buf[n] = ' ';
                        printf("%s", buf);
                }
                if (Sigurg) {
                        /* receive Out-of-Band data */
                        recv(Sock, &mark, 1, MSG_OOB);
                        printf("urgent message %c0, mark);
                        Sigurg = 0;
                }
        }
}

oobdata ()
{
        Sigurg++;
        nSigurg++;
}

sigquit ()
{
        printf("Quitting, %d sigurgs0, nSigurg);
        if (shutdown(Sock, 2))
                perror("shutdown");
        close(Sock);
        exit(0);
}
```

See also the following topics found in the *UNIX Interface Reference Manual, Inter-Process Communication Primer,* and *Tutorial Examples of Interprocess Communication in Berkeley UNIX 4.2 BSD* by Stuart Sechrest, Computer Science Research Group, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley.

accept(2)
bind(2)
close(2)
connect(2)
fcntl(2)
intro(2)
ioctl(2)
listen(2)
recv(2)
shutdown(2)
socket(2)
write(2)
signal(3)
intro(4)
inet(4F)
tcp(4P)
fcntl(5)
protocols(5)

Data Communications/Networking

le0 Errors on Sun 3/50's

The AnswerLine Service is receiving calls from Sun 3/50 owners who are seeing the errors shown below.

```
le0: Received packet with STP bit in rmd cleared
le0: Received packet with ENP bit in rmd cleared
```

Manual page `le(4s)` indicates that these errors occur in conjunction with a LANCE feature that the driver does not use. These errors can also occur in other circumstances described below. The manual page will be so revised for the 4.0 release of the Sun operating system.

Probable Cause

These errors are most likely caused by having a packet on the net that is too long. These error messages are probably generated when a Sun 3/50 gets a packet that is over the 1518 byte maximum length.

Detailed Explanation

We will use the definitions appearing below in this detailed explanation.

STP = STart of Packet
ENP = ENd of Packet

Under normal operation, the LANCE driver should not encounter receive descriptors when either the ENP or the STP bit is cleared.

The driver sets up buffers large enough for the maximum packet size allowed by the Ethernet specification. Chaining receive buffers so that an individual packet straddles multiple receive buffers is not necessary. Thus, there should be only one descriptor for each incoming packet. Further, this descriptor should have both the STP and ENP bits set.

It is possible, however, for an incoming packet to be too large to fit into a single receive buffer. This occurs when traffic on the net is in violation of the Ethernet specification. The incoming packet then spans multiple descriptors. The ENP bits are clear on all but the last descriptor and the STP bits are clear on all but the first descriptor. These conditions cause the two error messages shown above.

A third error message about babbling confirms the condition indicated by the first two error message. A memory access timeout error message may be caused when the driver becomes confused by the over-length packets and is not recovering properly. Two final error messages indicate that the chip has decided to stop its transmitter and receiver since net traffic is out of specification. The driver then restarts the transmitter and receiver upon receipt of these last two error messages.

Summary

It is likely that other equipment on the net is operating outside the Ethernet specification by sending over-sized packets.

Time Zones, Kernals, and You

UNIX Kernals and Greenwich Mean Time (GMT)

All UNIX kernals use GMT. However, some systems allow users to set environmental time variables as they like. This allows users to dial in from any time zone and have the date command reflect the correct local user time.

The time zone, TZ, variable is supported by any program built in the System V, release 3.2 environment. This includes all commands in `/usr/5bin`, which contains less than 1% of the commands. Sun supplies S5 versions of commands where there is a significant incompatibility between release 4.2 and S5 versions. Thus, 99% of the commands are located in `/bin`, `/usr/bin`, `/usr/ucb`, and the like. These commands are built with the release 4.2 libraries and do not understand the TZ environment variable.

Time Zone (TZ) Problems

There are two problems with the TZ environment variable. First, it is difficult to get TZ into every process' environment. Vanilla S5 systems attempt a solution by setting TZ in `/etc/rc` for programs and their children that run from there. TZ is also set in `/etc/profile` for login Bourne shells. This, however, omits any user with non-Bourne login shells. You can fix this for the C-shell by putting TZ in your `.login`, or by making a system-wide `.login` file (`/etc/csh.login`, for example) and placing TZ there.

However, this fix does not solve the TZ problem for some specialized applications where the login shells are not shells in the standard UNIX sense. You can fix TZ here by modifying the S5 login to preserve the TZ value in its environment and having something set TZ before running `getty`. Note that `init` does not run `getty` directly in S5. `init` has a table telling it what programs to run and when. Each line in the table contains a UNIX command to be run. For example, you could run the `env` command with arguments telling it to set TZ and then to run `getty`.

The standard S5 as distributed by AT&T uses Eastern Standard Time (EST) if TZ is not set. Vendors may change the time to their local time zone. Xenix continues to use the V7 `ftime` call, which gets time zone information from the kernel. You are not forced to set TZ in the environment to get a local time zone.

Second, some programs do not use the user time zone but need to know the time zone in which the the computer is located. An example is `uucico`, which makes long-distance calls to other machines after 2300 in the computer's time zone, not that of the user. A user can force the computer to make a long-distance call at any local time by setting TZ as required and then running `uucico`.

A History of Time

Prior to the V7 `ftime` call, V6 had neither an environment nor an `ftime` call. It supported the time zone complied into the C library only. If you were not in the same time zone as Dennis Richie's computer, you had to recompile `ctime` and rebuild every command that used the date.

V7, from AT&T and not Berkeley, fixed this by adding an `ftime` call that provided the current clock value, finer time resolution to include milliseconds, the local time zone offset from GMT, and a flag indicating whether Daylight Savings Time (DST) was to be used. You then merely had to reconfigure the kernel. Release 4.1BSD, based on UNIX/32V which is a VAX variation of V7, used `ftime` as well.

Release 4.2BSD improved `ftime` by replacing it with `gettimeofday`. This new call provided the time as `timeval`. It provided higher resolution in a standard, system-wide format, the GMT offset, and the DST flag. The flag now included what type DST was to be used. This accounted for the fact that not all countries go on and off DST at the same time as the United States. Further, the dates for the beginning and end of DST differ as well. Release 4.2BSD provided tables for some locations outside the USA, but those tables were not always correct. Some fixes were made in release 4.3BSD. Sun OS release 3.2 provides additional fixes correct at the time of release.

System III did not include `ftime`. It sets the current time zone from the TZ environment variable. This arrangement has the same problems detailed above and it still does not cover time zones outside the USA. Problems arise when the USA, Australia, and Europe did not go to DST at the same times.

Release 4.2BSD is somewhat more accurate, although it does not understand Australian and European rules completely. Canada is treated as being under the same rules as the USA and the USA DST changes of 1987 are not incorporated. Release 4.3BSD has corrected some of the European and Australian rules, as well as Canadian rules that did not change in the mid 1974-5.

Time and Time Again: Sun OS Release 3.2

Sun OS Release 3.2 has rules that should be correct for Europe. The Australian rules have two variants. The first starts on the last Sunday in October and ends on the first Sunday in March. The second does the same thing until 1985 when it ends on the last Sunday before March 21 and in 1986 when it starts on the last Sunday in October and ends on the last Sunday before March 21. According to the Australian consulate, DST starts on the next-to-the-last Sunday in October and ends on the last Sunday before March 21. Finally, Sun OS Release 3.2 includes USA changes made in 1987.

Sun OS Release 3.2 should also choose the proper time to start and end DST. It starts at 0200 standard time and ends at 0200 DST in the USA and Canada. It starts at 0200 standard time and ends at 0300 DST in Australia. It starts at 0100 standard time and ends at 0200 DST in Great Britain, Eire, and the Western European time zone. It starts at 0200 standard time and ends at 0300 DST in the Central European time zone.

Note that these latest tables are compiled into the `ctime` code. Only applications built with the Release 3.2 library will use them. Applications built with earlier library versions will give the same results under Release 3.2 that they gave under earlier releases.

There is no facility in Release 4.2 to do the same time calculations. However, Arthur Olson at the US National Institutes of Health (NIH), implemented a new version of the time zone code that permits four things:

1. Permits you to set TZ to alter the time zone information you see.

2. Permits programs like `uucico` to see the correct local time zone, regardless of the TZ value.

3. Permits you to set up any time zone conversions in accordance with the wisdom of your local politicians.

4. Permits you to change time zone conversions as political wisdom changes, as it is want to do from time to time.

The TZ Environment Variable

The TZ environment variable is treated like a filename. The file contains a DST rules table. Given the filename, a routine reads in the table. If the table has not already been read in, `ctime`, `localtime`, and the S5 routine `tzset` call this routine with the value of the TZ environment variable as the filename. The routine looks for the file named `localtime` if a null pointer is passed. You will then get the proper local time if TZ is not set or if the program explicitly calls this routine with a null pointer like `uucico` does.

This will probably appear in Release 4.4BSD (date unknown) and may appear in a future Sun OS release. This would free developers from having to anticipate future DST rules changes.

Questions and Answers

Chuq's Hints & Tips #1

I have created this column for two purposes.[1] First, some questions are asked regularly on the Technical Support Hotline. I feel everyone can benefit from distributing discussions of these problems as widely as possible. Second, a large and constantly growing body of information, hints, and tips are not documented anywhere.

I will collect and distribute these information nuggets in this continuing column so that we can all learn from them. I will cover unusual topics, but this column should not be used as an alternative to contacting your salesperson or using the Hotline.

If you have a question that you would like answered in this column, please mail your question to "Software Technical Bulletins" at Sun Microsystems, Inc., 2550 Garcia Avenue, M/S 2-34, Mountain View, CA 94043. You can also send in your question by electronic mail to *sun!stb-editor*. (Remember to call Sun Customer Software Services AnswerLine at 800 USA-4-SUN for technical questions on this column or any other article in this bulletin. I look forward to hearing from you!

I will answer selected questions in each Software Technical Bulletin issue. Also, if you have hints or tips you want to share, feel free to mail them to *folklore%plaid@sun.com* or to *sun!plaid!folklore*. I will be glad to receive them.

Using the /etc/group File - Release 3.2 and Beyond

The /etc/group file is processed differently starting with release 3.2. When you log in, the login program searches /etc/passwd and /etc/group. It may also search associated yellow pages (YP) maps, if applicable, for groups you should be in. You can be in up to eight groups with your primary group being designated in the/etc/group file.

In release 3.2, the routines doing this search were significantly reworked. They are now less tolerant of bad data in the file. A file that works fine under release 3.0 may mysteriously fail under release 3.2. This failure is silent with no error messages displayed. The only way for you to see the problem is with the "groups" command. You will see that you are not in some of the groups you are supposed to be in.

In releases through 3.0, errors in the /etc/group file and associated YP maps were ignored. Starting with release 3.2, however, the routines now stop searching the file and return when they see a problem. Any entries after the incorrect one are not read and those groups will not be added to your environment.

[1] This continuing column is submitted by Chuq Von Rospach, Customer Software Services.

The situation is more complicated on the YP master server. If you include the "+:" in the server /etc/group file, it will be installed in the YP group.byname and group.bynumber maps. However, when you try to read through the YP maps, it is considered a normal, albeit malformed, group entry. This causes the group search to terminate.

The "+:" can come out anywhere in the group lookup even if it is at the bottom of the server file. This is due to the randomizing effects of the way YP databases are stored. To illustrate, do a ypcat group and you will notice that the printing has nothing to do with the way data are stored in the ascii file. Anything in the group maps appearing after this bad "+:" entry in the ypcat listing will neither be read by the group routines nor installed in your environment.

The Fix

The fix for this situation is to go to the YP master server and remove the "+:" from the bottom of the /etc/group file. It is not necessarily on that machine since it has all available groups by definition. It then does not have to do YP lookups. This is another case of something that worked under release 3.0 that fails under release 3.2.

Also, check for blank lines in the /etc/group file. These were ignored under releases through 3.0, but cause the search to end under release 3.2. Fortunately, a new program called 'grpck' is available in the optional System V software. This program reports any problems in your group file. It will help you track down any YP maps problems if you run it on your master server.

You can check your group file even if you do not have the optional grpck program installed. Use the vi command shown below on a COPY of the group file since it is destructive and you do not want to accidently delete your data.

```
:%s/.*:.*:.*:.*//
```

This command will delete all properly formatted group file lines. Anything left over is missing a colon, the probable cause of your problem. Please note that any fixes you make to the group file or maps will not be recognized until the user logs off and on again since these data are set up once upon login.

Tip of the Month - (TOM)

Under *suntools*, it is possible to reprogram the *shelltool* and *cmdtool* namestripes. The cshell aliases shown below see if you are running *suntools*. If yes, it sets things up so your directory stack will be displayed in the namestripes. If no, it modifies your prompt to show your hostname and current working directory. It then modifies the cd, pushd, and popd commands to keep the namestripes up to date.

Please note that the sequence "^[" is an escape character and not two separate characters. In vi, type `^v^[` for the escape character.

```
set tty = `tty`
set hostname = `hostname`
if (${tty} != /dev/console && $?term &&
  ($term == sun || $term == sun-cmd)) then
```

```
    alias hdr  'echo -n "^[]l*^[
    alias ihdr 'echo -n "^[]L*^[
    alias setbar 'hdr "${hostname}: `dirs`"; ihdr $cwd:t'
    set prompt = "${hostname}% "
else
    alias setbar       'set prompt = "${hostname}:$cwd% "'
endif

alias .        'dirs;setbar;jobs'
alias cd "cd * ; setbar"
alias pushd "pushd * ; setbar"
alias popd "popd * ; setbar"
setbar
```

4

IN DEPTH

4

IN DEPTH

The Yellow Pages Service

The Purpose of the Yellow Pages (YP) Service

The Yellow Pages (YP) Service provides a set of maps or data files common to one or several systems. These maps contain network configuration information about these systems. Workstations and terminals then use these maps instead of having files of their own.

The yellow pages server contains a set of yellow pages maps. YP clients bind to the server to access these maps. Only the server maps need updating since it sets up a domain to which YP slave servers and clients belong. Multiple YP domains may exist on the same network.

The YP server makes its maps from configuration files used in Ethernet networking with other systems. A YP client checks its local file first. Not finding the information locally, it then consults the YP server maps. The configuration files used to create the maps are shown below.

```
/usr/lib/aliases
/etc/ethers
/etc/hosts
/etc/ethers
/etc/group
/etc/netgroup
/etc/networks
/etc/passwd
/etc/protocols
/etc/services
```

Customers get the best use of the yellow pages service at a site including large numbers of computers which are networked together in a common network. Such sites are constantly dynamic, adding or moving systems, and adding or removing users from the networked systems. A number of Sun workstations may be included which network with each other and other computers through utilities including `rlogin`, `rsh`, `telnet`, `ftp` and `tftp`.

System administrators may have difficulty maintaining the several network configuration files for each system when the environment is constantly changing. The yellow pages service helps by having only one set of master maps that is changed by the master YP server for all Suns workstations on the network.

Other workstations use the master maps by binding to the YP server. This bind process runs continually on the YP client, the workstation being served, and is called `ypbind`. The YP server runs `/etc/ypserv` and `/etc/ypbind` to serve its YP clients.

Once a day or week the system administrator updates the YP master server `/etc/hosts`, `/etc/others`, `/etc/passwd`, and other files. The system administrator then updates the yellow pages by remaking the YP maps. This procedure takes only a few minutes. The YP clients then bind to the YP server and use the updated maps to find host- and user-names only after the client determines that its local files do not have the needed information.

A single YP server may not be sufficient to meet client YP requests in a large network environment. Sun yellow pages have YP slave servers configured in such large networks. These YP slave servers access YP maps and clients bind to these slave servers using `/etc/ypserv` and `/etc/ypbind`. To further aid the customer, YP slave server configuration files are updated automatically, receiving the updated YP maps from the YP master server.

More than a single YP master server is found in very large networking environments. You may find multiple domains on these large networks since each YP master server defines a separate YP domain. There is no conflict since each YP clients knows which domain it is in. The YP client's domain name and YP slave server hosts are determined when the YP master server is configured by `setup`. The `/bin/domainname` utility returns the YP domain name to which a YP client or slave server belongs.

YP Server Installations

The YP master server, slave server, and client system is installed using the steps listed below.

1. The system administrator initially runs `setup` to install the Sun system. The YP master server is configured as such at this time. The names for all possible YP slave servers are also configured on this same machine. The names are also configured on the YP slave servers at this time.

2. Update the YP master server YP maps daily by changing the YP master server configuration file or files (`/etc/hosts`, `/etc/passwd`, and the like), and by running `/etc/ypmake <mapname>` or `/etc/ypinit` for all maps as needed.

3. The YP master server then sends these updated YP maps out to the YP slave servers. They should be running `/etc/ypserv` to automatically receive the updated maps. The system administrator otherwise has to do a `yppush`

to the slave server when it is ready.

4. The YP client then can run `/etc/ypbind` which is broadcast to all YP slave servers. The slave server responding first is the one to which the YP client is bound. This process assumes that the YP slave server responding first is the least busy so that the networking load is most evenly distributed. This process continues and the client stays bound to the same slave server until the YP client reboots.

5. The YP client automatically rebinds to another YP server in the event a YP server goes down. The YP client requests a YP service to its YP server via an `rpc` call. The server may or may not respond. The client drops its `ypbind` with this server and `ypbinds` to another server if a second request results in no response.

6. For example, the YP clients runs a request for telnet host56832. First, the YP client's own `/etc/hosts` file is consulted. If no entry for host56832 is found, the YP client requests a service for a YP map lookup using an `rpc` call to the YP server. This request for the internet host56832 address goes to the client's YP slave server or to the master server if no slave server exists. The YP server then responds with the internet address. The YP client continues and performs a telnet request to host56832.

YP Utility Summary

A selected list of YP utilities appears below.

1. `/etc/ypserv`

   The process the YP master or slave server runs to serve YP maps.

2. `/etc/ypbind`

   The process any YP master or slave server or client runs to issue or resolve YP requests via `rpc`.

3. `/etc/ypwhich`

   Returns the YP service domain name.

4. `/etc/ypwhich -m`

   Returns which YP master or slave server owns the YP maps. The one which ran `ypmake` is the one who made the new map. Check this if you suspect that your YP maps are incorrect. It may be the wrong server who ran the `ypmake`.

5. `/etc/yppoll`

   To find out the version of the YP maps you are using. This command may be run on any machine, YP server, or client.

6. `/bin/domainname`

   Returns the YP domain name that the system is in. This is set in the /etc/rc.local file on each system whether a YP server or client. This should be checked if the YP maps appear incorrect.

7. `/etc/ypcat`

   Does a cat of a YP map. For example: "/etc/ypcat hosts" should provide a list of the hosts known to YP.

8. `/etc/ypmatch`

   This utility is new to release 3.0 and provides a grep-like feature. For example, "/etc/ypmatch joe passwd" should show a passwd entry known to YP.

Revision History

Revision	*Date*	*Comments*
DRAFT 1	April 1987	Third issue of Software Technical Bulletin (Software Information Services).

Corporate Headquarters
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
415 960-1300
TLX 287815

For U.S. Sales Office locations, call:
800 821-4643
In CA: 800 821-4642

European Headquarters
Sun Microsystems Europe, Inc.
Berkshire House
High Street
Ascot, Berkshire SL5 7HU
England 0990 28911
TLX 846573

Germany: 089 416-00820
UK: 0990 25942
France: 01 630 23 24

Canadian Headquarters
416 477-6745

Europe, Middle East, and Africa, call European Headquarters:
0990 28911

Elsewhere in the world, call Corporate Headquarters:
415 960-1300
Intercontinental Sales