Porting to Solaris 2.x

Ron Winacott
Marc Staveley
Georg Nikodym

(ronw@canada.sun.com)
(marc@staveley.com)
(georgn@canada.sun.com)

Revision: B.4

Table Of Contents

Introduction

- OpCom & Services

OpCom & Services

OpCom:

To support Key SMCC End Users, OEM's and ISV in their migration to Solaris 2.x. We provide in-depth software support, porting and migration advice, and support for several early access programs.

News Letter:

SunOpsis.

OpCom ftp server:

Internet server - opcom.sun.ca

Other courses:

Solaris Device drivers and Streams writing.
User Level Threads for Programmers

NIS+ for Administrators

Consulting Specials:

SCP (Stand-alone Copy Program)
Configurable Network Boot
TCP Connection Caching

To get the news letter, send mail to solaris2@sun.com requesting to be added to the mailing list, or check the ftp server.

The internet FTP server is open to anyone, login as ftp, passwd your email address.

OpCom also offers other detailed courses,

and

consulting specials for Solaris 2.x

SCP allows creation of a bootable image of a master disk (OS and application(s)) on a variety of media formats

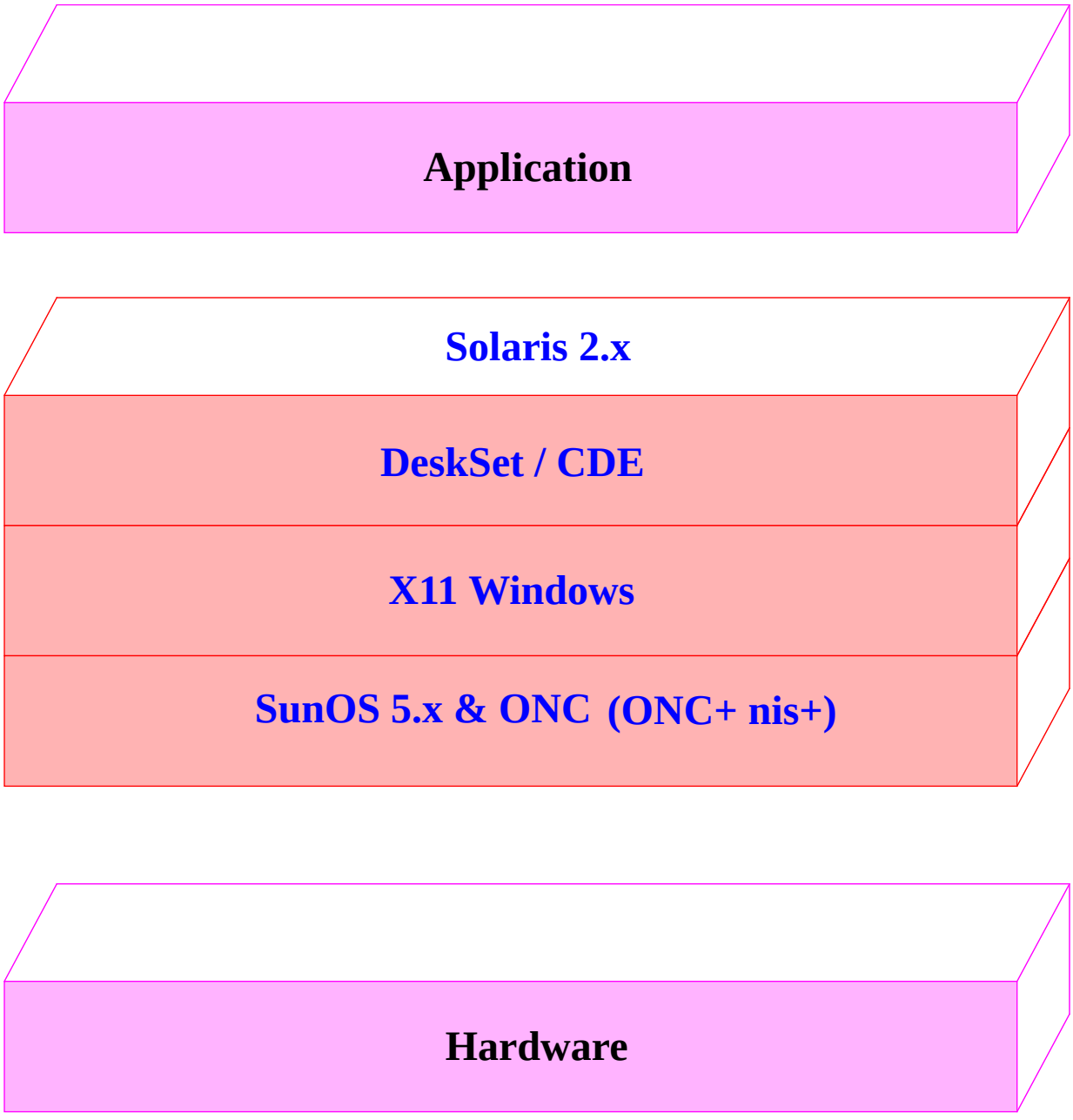
Configurable Network Boot supports a single install server on a complex network

TCP Connection Caching caches socket connections between process to lower open setup time (i.e. improves performance in applicable cases)

Solaris 2.x Overview

- What is Solaris 2.x?
- What is SunOS 5.x?
- /devices & /dev
- /proc
- Libraries
- Packages
- Name Service

What is Solaris 2.X?



The One and only marketing slide!

Solaris is a complete operating environment not just an OS.

ONC+ = TI-RPC, NFS, NIS+

What is SunOS 5.x?

Dynamically loaded,

fully preemptable,

symmetric multiprocessing, threaded, kernel

with

time-sharing and real-time scheduling classes

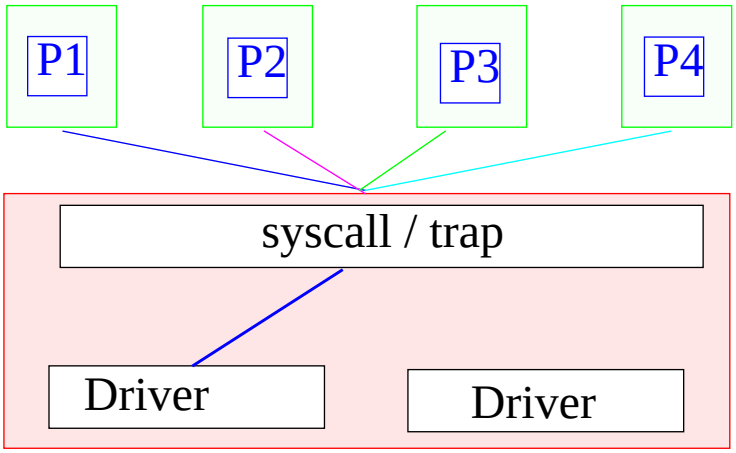
supporting multiple threads of control

within a single application process.

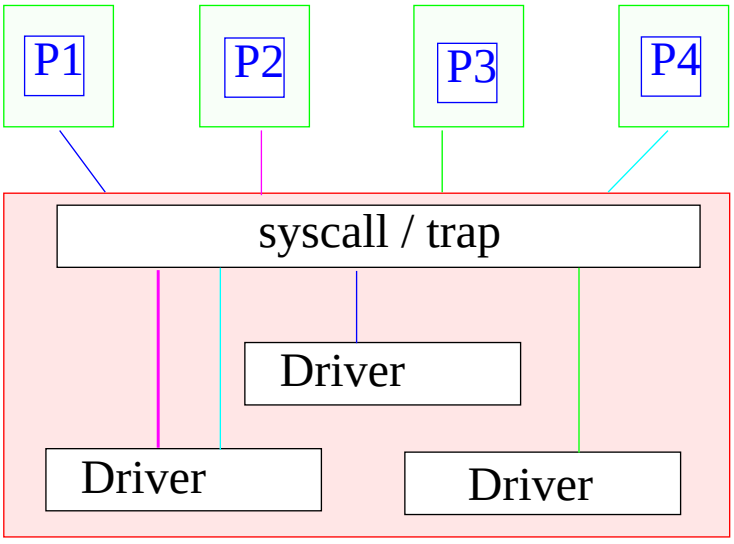
And it does windows too.

Symmetric Multiprocessing & Multi-Threaded kernel

4.x on MP



Solaris 2.x on MP



All CPUs are equal, and have the same view of the world.

All the CPUs could be running kernel code.

All sharing the same bus, memory, etc...

as opposed to

asymmetric = master / slave CPUs.

The *“kernel thread”* is now the scheduled entity, not the process.

With a threaded kernel, the system can have more then one thread running simultaneously.

Which allows more then one system call to be running at the same time.

Kernel daemons like “biod” now run on *kernel threads*.

Interrupts are run on kernel threads and are scheduled like any other process, no more need for special code.

Fully Preemptable.

The Solaris kernel does NOT have preemption points where the state of real-time or higher priority threads are checked.

A runnable thread can preempt the running thread if it has a higher priority anywhere and anytime in the kernel.

This kernel structure will give a real-time thread a dispatch latency of about 2 milliseconds on a SPARCstation 1.

Stock SVR4 has preemption points in the kernel about every 100ms spacing.

Kernel preemption is turned off in critical sections of code. These sections are very short, and few.

Dispatch latency is the time taken to move the highest priority process from the dispatch table to a CPU.

(TS_RUN to TS_ONPROC)

Dynamically loaded.

/kernel/unix is a small base file.

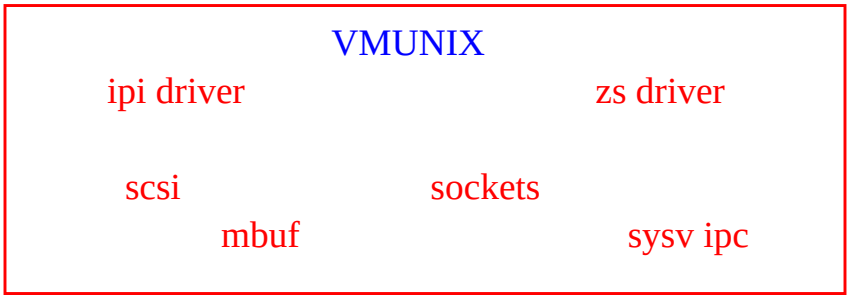
All the other drivers, streams, file systems, system calls, scheduling classes, etc...

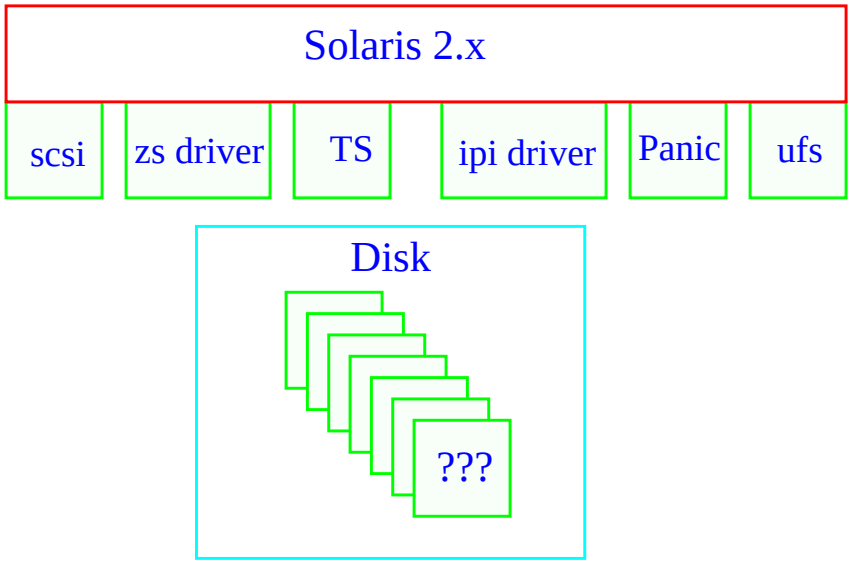
are loaded into the system when they

are needed. They are also unloaded when

they are no longer needed to save kernel resources

(at the discretion of the implementor).





/kernel/genunix and /platform/$ARCH/kernel/unix in 2.5

/vmunix had to be rebuilt each time you added a device driver, or needed to tune system parameters. This is no longer true.

No more need for a bundled “C” compiler!

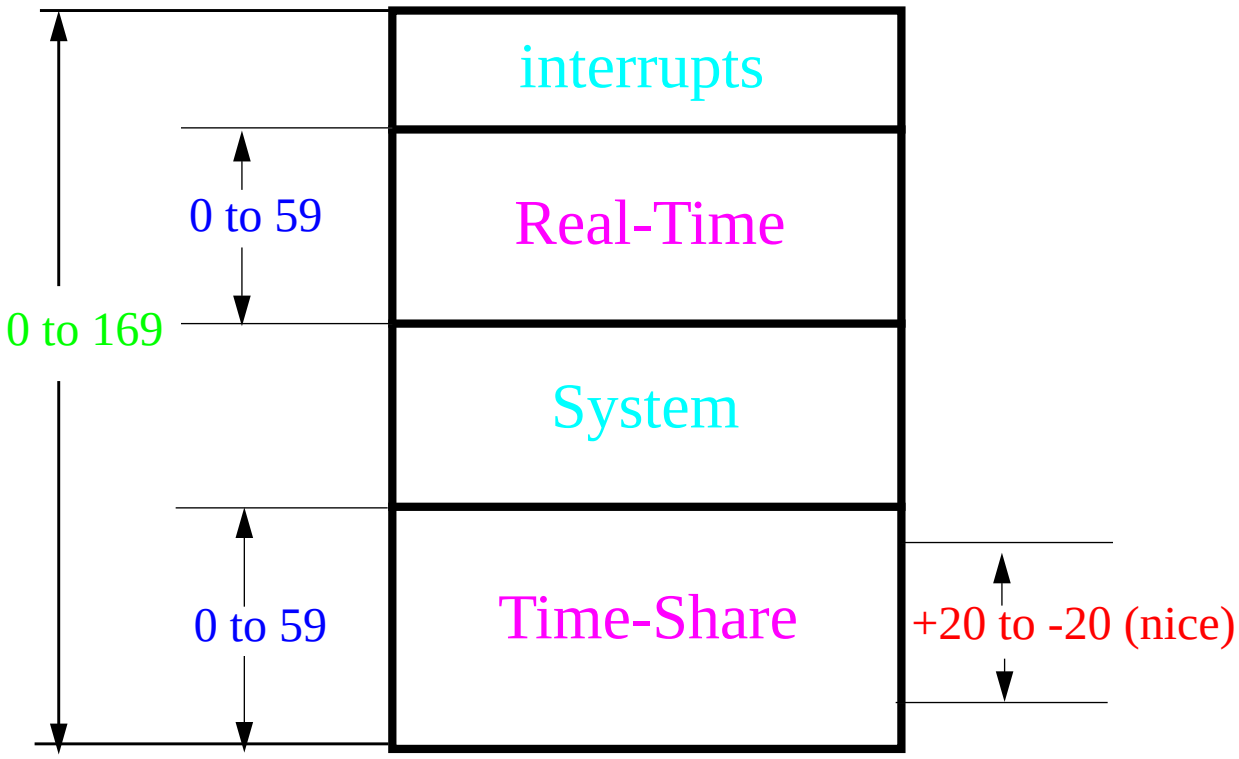
There were no loadable system calls, and limited loadable module support in SunOS 4.x.

Solaris now stores the kernel in a subdirectory, not in root. All the loadable modules are also stored in subdirectories in /kernel/* and /usr/kernel/*. In Solaris 2.5, /platform/$ARCH/* and /usr/platform/$ARCH/* hold the platform specific modules.

Note: This means that the KERNEL can open, and read file systems, directories, and files. (the kernel has context)

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

_fini()
{
   if (module_reffcnt!= 0)
        return (EBUSY);

   return (mod_remove(&modlinkage));
}
```

Time-sharing / Real-time



The dispatch table.

Table driven priorities! (see dispadmin(1m)).

Not:

$$p\_cpu = p\_cpu + 1$$

(once per tick on current process)

$$p\_cpu = \frac{2 \times load}{2 \times load + 1} \times p\_cpu + p\_nice$$

(once per second for all processes)

$$p\_usrpri = PUSER + \left[\frac{p\_cpu}{4}\right] + 2 \times p\_nice$$

(once every 4 ticks per runnable process)

Global priorities from 0 to 169, 0 being lowest.

Time share GLOBAL priority from 0 to 59.
Kernel (sysclass) priority from 60 to 99.
Real-time (if loaded) from 100 to 159.
Interrupts (always the HIGHEST) from 160 to 169
(if RT is loaded, else from 100 to 109).

Each scheduling class has its own priority range.
i.e. the highest RT priority to the user is 59! The kernel will add the offset (100 for RT).

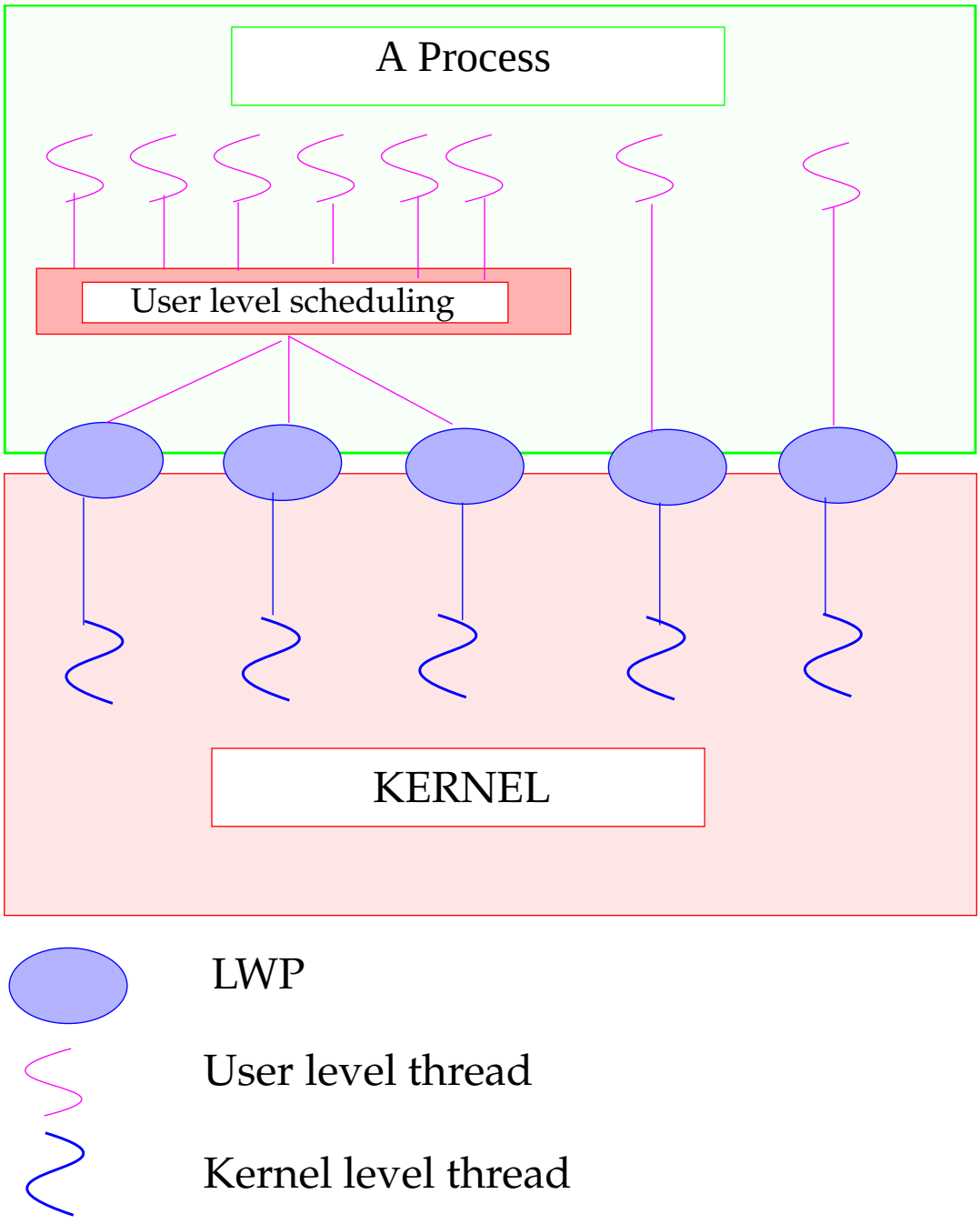
RT = External event driven, fixed priority.

Interrupts are scheduled like any other process, they do not “steal” the context of the interrupted process. As a result, you can have more then one pending interrupt of the same type.
(up to five)

Callout() is run at the priority of interrupt 10 (tick). The callback function is run at the priority of the requesting thread.

OpCom wrote two new scheduling classes,
BT (batch) and FX (fixed).

User level Threads

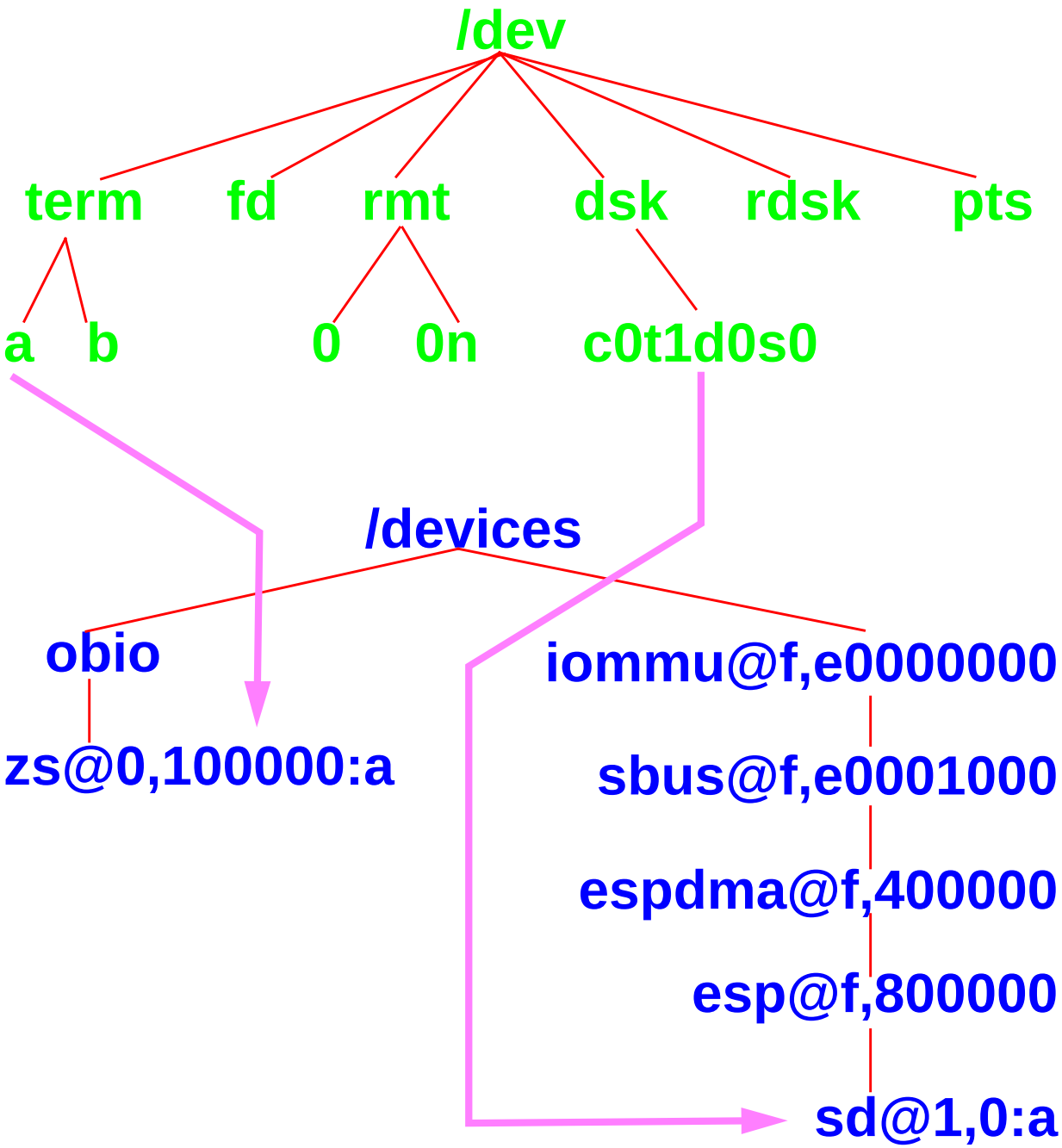


With user level threads, the application programmer can increase the amount of concurrency within a program.

Using threads, there is no longer a system call bottleneck within a process;
blocking operations no longer block the process, just that thread.

Remember *kernel thread* is now the scheduled entity, not the process.

/devices & /dev


/dev
term
fd
rmt
dsk
rdsk
pts
a
b
0
0n
c0t1d0s0
/devices
obio
zs@0,100000:a
iommu@f,e0000000
sbus@f,e0001000
espdma@f,400000
esp@f,800000
sd@1,0:a

/dev is now Hierarchical.

New names for most devices:

/dev/ttya	=>	/dev/term/a
/dev/sd1c	=>	/dev/dsk/c0t1d0s2
/dev/rsd0a	=>	/dev/rdsk/c0t0d0s0
/dev/st0	=>	/dev/rmt/0

There are no character or block special files in /dev, all the files in /dev are symlinks to /devices/xxx

(linkage rules are defined in /etc/devlinks.tab).

If BCP is installed, the old SunOS 4.x names are created in /dev (as symbolic links to the new names).

The device tree is built by the Open Boot Prom (OBP). This tree is passed to the kernel at boot time allowing the kernel to create the /devices tree.

Slot assignment and address assignment is dynamic.

To rebuild /devices, reboot the system using the `boot -r` prom command with all the devices turned ON! The device nodes in the /devices tree are created by the kernel at the request of the driver.

NEVER DELETE, CHANGE, ADD ANYTHING IN /devices

/proc

/proc file list (process list)

```
-rw------- 1 ronw    4132864   Apr 14 20:24  02949
-rw------- 1 ronw    905216    Apr 14 20:24  02951
-rw------- 1 ronw    655360    Apr 14 20:26  02961
-rw------- 1 ronw    1417216   Apr 14 20:26  02962
-rw------- 1 ronw    1327104   Apr 15 13:17  03358
-rw------- 1 ronw    0         Apr 15 13:17  03359
-rw------- 1 ronw    909312    Apr 15 19:48  03674
-rw------- 1 ronw    10367488  Apr 15 19:48  03675
-rw------- 1 root    1622016   Apr 16 11:03  03784
-rw------- 1 ronw    3948544   Apr 16 13:36  03868
-rw------- 1 root    2519040   Apr 16 13:44  03881
-rw------- 1 lisat   851968    Apr 16 13:44  03882
-rw------- 1 lisat   4210688   Apr 16 13:44  03890
-rw------- 1 ronw    655360    Apr 16 18:38  03951
```

see the *proc(4)* man page for more programming info

see the *proc(1)* man page on 2.5 for information on the proc tools

A flat file system (procfs) that holds all the process ID’s of all the running processes. (invented at Bell Lab’s for UNIX v8.)

You can use open, close, lseek, read, (write) on a “file” in the /proc directory. You are opening, reading, etc, in the processes address space.

The new dbx, and debugger use this file system to get the information on any process you are debugging.

All the processes (files) use the same file permission semantics that normal files do.

Why the size zero on pid 03359?

Libraries

Library listing

/usr/lib/ld.so		/usr/lib/libnsl.so	*
/usr/lib/libadm.so		/usr/lib/librac.so	
/usr/lib/libadmagt.so		/usr/lib/libresolv.so	
/usr/lib/libadmapm.so		/usr/lib/librpcsvc.so	
/usr/lib/libadmcom.so		/usr/lib/libsocket.so	*
/usr/lib/libadmsec.so		/usr/lib/libsys.so	
/usr/lib/libaio.so		/usr/lib/libthread.so	*
/usr/lib/libc.so		/usr/lib/libthread_db.so	*
/usr/lib/libelf.so		/usr/lib/libw.so	
/usr/lib/libgen.so	*	/usr/lib/nss_compat.so	X
/usr/lib/libintl.so		/usr/lib/nss_dns.so	X
/usr/lib/libkrb.so	*	/usr/lib/nss_files.so	X
/usr/lib/libkstat.so		/usr/lib/nss_nis.so	X
/usr/lib/libkvm.so		/usr/lib/nss_nisplus.so	X
/usr/lib/libm.so		/usr/lib/straddr.so	X
/usr/lib/libmapmalloc.so		/usr/lib/switch.so	X
/usr/lib/libnisdb.so		/usr/lib/tcpip.so	X

* = libraries of interest.

X = dlopened at runtime for TI, and name service switch.

New libraries.

Some of the functions that were in libc in SunOS 4.x have been moved to other libraries.

Changes to your Makefiles will have to be done to link the correct libraries.

ld is good at telling you what you are missing.

eg.

nlist(3E) - libelf.so

regexp(3G) - libgen.so

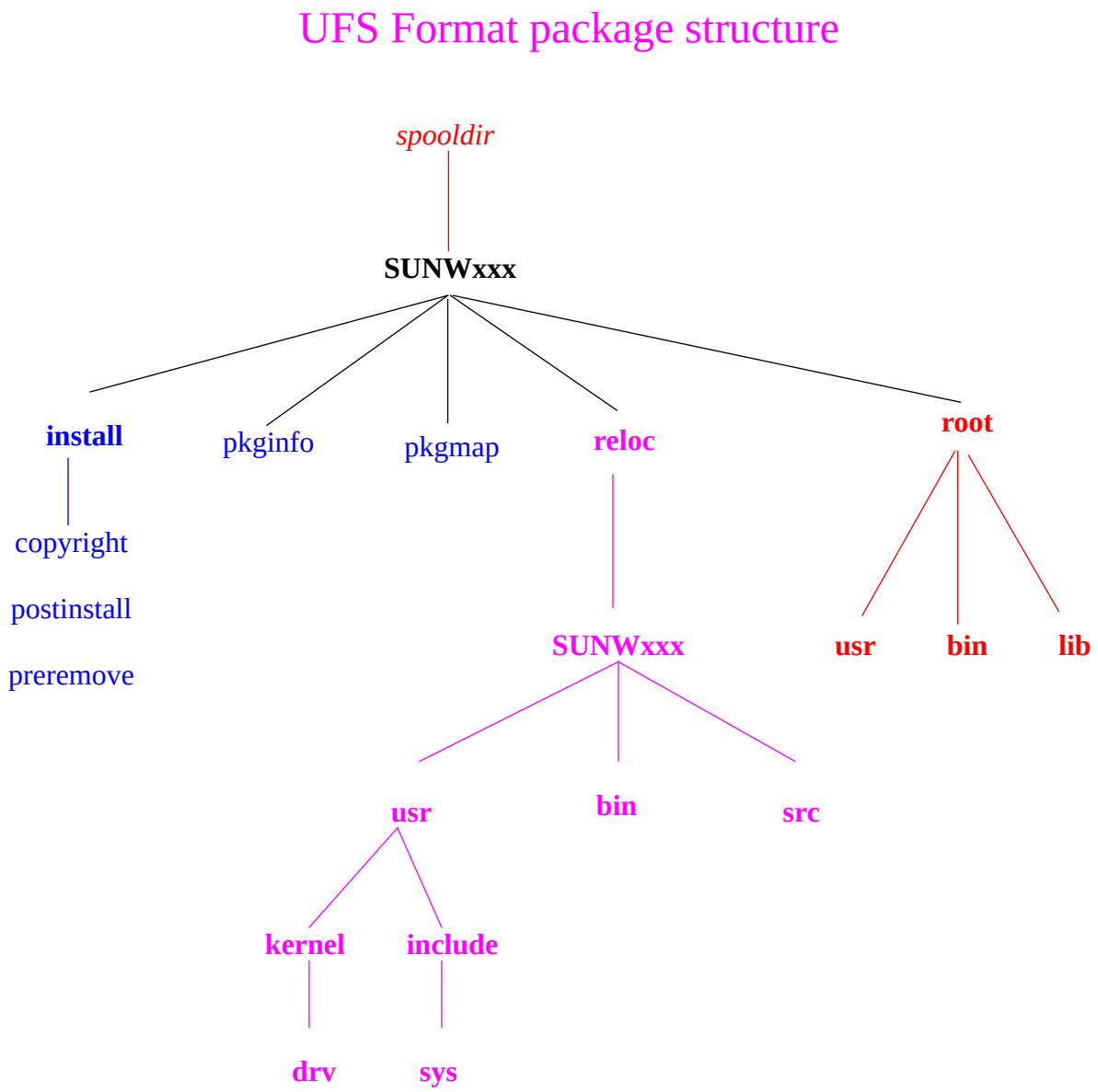
connect(3N) - libsocket.so

gethostbyname(3N) - libnsl.so

See "*whatlib*" on the ftp server.

Also "*xref_report*" in the `OPCOMxref` package.

Packages


UFS Format package structure
spooldir
SUNWxxx
install
pkginfo
pkgmap
reloc
root
copyright
postinstall
preremove
SUNWxxx
usr
bin
lib
usr
bin
src
kernel
include
drv
sys

All software is now installed using the pkgadd(1) command.

To build a package:

Name your software package, 14 char. limit. (SUNWxxxx)
Organize the package contents.
Create the prototype file. (pkgproto)
Create the package info file.
Create an on-disk version using pkgmk (the result is pkgadd’able)
May be converted to stream form (ie. tape) with pkgtrans.

Installing a package:

pkgadd <-d directory>

Removing a package:

pkgrm package_name

Things under the “root” directory in the package are anchored at “/”, and things under “reloc” are anchored at “$BASEDIR” (from pkginfo file or /var/sadm/install/admin/default)

Name Service.

Use of the /etc/nsswitch.conf file:

```
#
# /etc/nsswitch.conf:
#
# example file that could be copied over to /etc/nsswitch.conf; it
# uses NIS+ (NIS Version 3) in conjunction with files.
#
# “hosts:” and “services:” in this file are used only if the
# /etc/netconfig file contains “switch.so” as a nametoaddr
# library for “inet” transports.
#
# the following two lines obviate the “+” entry in /etc/passwd
# and /etc/group.
passwd: files nisplus
group: files nisplus

# consult /etc “files” only if nisplus is down.
hosts: nisplus [NOTFOUND=return] files

# Uncomment the following line, and comment out the above,
# to use both DNS and NIS+
#hosts: nisplus dns [NOTFOUND=return] files

services: nisplus [NOTFOUND=return] files
networks: nisplus [NOTFOUND=return] files
protocols: nisplus [NOTFOUND=return] files
rpc: nisplus [NOTFOUND=return] files
ethers: nisplus [NOTFOUND=return] files
netmasks: nisplus [NOTFOUND=return] files
bootparams: nisplus [NOTFOUND=return] files

publickey: nisplus

netgroup: nisplus

automount: files nisplus
aliases: files nisplus
sendmailvars: files nisplus
```

Easy name server configuring.

nsswitch can use more then one name service provider.

NIS+, NIS, files, and DNS

There are nsswitch.{files,nis,nisplus} supplied.

Sun install will copy the correct file over.

An application may work in your environment but **not** at the customer's site if their `nsswitch.conf` uses a different name service supplier that has **different semantics** than the one at your site.

E.g. `getpwent()` may return different information when using NIS+, NIS, or files.

Standards

- ANSI ’C’
- IEEE POSIX
- SVID
- X/Open
- Implentation Specific
- IETF RFCs

ANSI X3.159-1989 - ‘C’

Also ISO/IEC 9899

Solaris 2.x header files and library functions are ANSI ‘C’ compliant.

Prototypes exist for all library functions.

Header files check for
#if defined(__STDC__)

Strict ANSI ‘C’ conformance is a good answer to the question “What can I do to make my programs more portable”[1]

In the ANSI ‘C’ header files non-ANSI prototypes and macros are elided if __STDC__ == 1

This can cause problems when using a compiler that sets __STDC__ = 1, the prototypes for non-ANSI functions that are in libc will not be included in the compilation.

In Solaris 2.4 and up, you may define __EXTENSIONS__ to include prototypes for non-ANSI functions even when __STDC__ == 1.

TABLE 1. ANSI ‘C’ header files

assert.h	float.h	math.h	stdarg.h	stdlib.h
ctype.h	limits.h	setjmp.h	stddef.h	string.h
errno.h	local.h	signal.h	stdio.h	time.h

1. Donald Lewine, “POSIX Programmer’s Guide”

Compiler Flags

The new SunSoft compiler allows both old and new style C code. The -X{s,t,a,c] options provide varying degrees of compliance to the ANSI ‘C’ standard.

TABLE 2. SparcCompiler “cc” options, and modes

Compiler option	__STDC__	Mode
-Xs	Undefined	Pre-ANSI, K&R C (like 4.x cc)
-Xt	0	ANSI Plus K&R. Favors K&R
-Xa	0	ANSI Plus K&R. Favors ANSI
-Xc	1	ANSI C only (conforming)

Note the setting of the feature test macro __STDC__

- -Xs - Senescent mode, accept all (pre-ANSI) K&R ‘C’ syntax. This is the old 4.x cc behavior. But is not 100% SunOS 4.x *cc* behavior. **This is <u>NOT</u> porting your code. Don’t use this option.**

- -Xt - Transition mode. If the semantics are different between ANSI ‘C’ and K&R ‘C’, the compiler will print a warning and use the K&R ‘C’ semantics. This is the default -X mode in SparcWorks 3.0.

- -Xa - If the semantics are different between ANSI ‘C’ and K&R ‘C’, the compiler will print a warning and use the ANSI ‘C’ semantics (the opposite of -Xt). This is the default -X mode in SparcWorks 4.0

- -Xc - Conforming mode. Only ANSI ‘C’ semantics are used. The compile will reject any non-ANSI ‘C’ constructs.

- -v - Causes the compiler to perform more and stricter semantic checks.

Type Qualifiers

volatile

```
sig_atomic_t volatile hadsig = 0;

handler() {
        hadsig = 1;
}

func(void) {
        signal(SIGTERM,handler);

        while (!hadsig) {
                ...
        }
}
```

const

```
char const c
        Can't change the contents of c.
char const * foo
        Can't change the contents of the
        string.
char * const foo
        Can't change the pointer.
        Can change the contents.
char const * const foo
        Can't change the pointer.
        Can't change the contents.
```

Volatile means “*exact semantics*”, i.e. take no code generation shortcuts when accessing such an object.

Allows the compiler to safely use optimization strategies without fear of references being kept in registers or removed.

Const means “*read-only*” values or pointers that do not change.

String literals may be “char const []” not “char []”. [1]

SunOS:

```
mktemp( "/tmp/fooXXXXXX" );
```

Solaris:

```
char name[BUFSIZ];
strcpy( name, "/tmp/fooXXXXXX" );
mktemp( name );
```

1. at the discretion of the compiler writer.

Prototypes

Will this code compile?

Will it run?

```
#include <string.h>

int
main( void )
{
        char line[80];
        char const *string = "This is junk, "
                             "and unused";[1]

        strncpy( line, string, 10 );
        line[10] = '\0';

        fprintf( "%s\n", line );
}
```

1. String concatenation

int fprintf(FILE *, const char *, ...);

Function prototypes allow checking of the number and type of parameters.

Prototyping would have found this problem.

Put prototypes in a *.h file and include the header file in both the function definition and function invocation files.

All *.c files should include:

- stdio.h (standard I/O) or stddef.h (NULL, size_t)
- stdlib.h (ANSI libc functions)
- unistd.h (Unix libc functions)
- string.h (string functions)

Promotion

What's wrong with this code?

foo.c:

```
foo( float j, float t )
{
        printf( "%f %f\n", j, t );
}
```

main.c:

```
main( void )
{
        foo( 0.0, 1.1 );
}
```

In main.c at the call to foo() the constants 0.0, and 1.1 will be promoted to type “double” (64 bits), since the function invocation is in the absence of a prototype. The function foo() will dereference the arguments as type “float” (32 bits).

The output will be: 0.000000 0.000000

A prototype in main.c for foo() will fix the problem.

There are a number of ways to find this problem:

- Using SunSoft “cc” with the ‘-v’ option and looking for *implicit function declaration* errors.
- Using "lint" and looking for *implicit function declaration* errors.
- Using a ‘C++’ compiler and looking for *undefined function* errors.
- Using "gcc" 2.x with the ‘-Wstrict-prototypes’ option to do strict prototype checking (this is not included in ’-Wall’).

Promotion (again)

prototypes.h:

```
extern int function( char c, short s );
```

function.c:

```
#include "prototypes.h"

function( c, s )
        char   c;
        short  s;
{
        .....
}
```

Must be careful when mixing K&R and ANSI styles.

This will produce 3 errors:
1 function redefinition error
2 prototype mismatch errors.

Should be:

prototypes.h:

```
extern int function( int c, int s );
```

function.c:

```
#include "prototypes.h"

function( c, s )
        char   c;
        short  s;
{
        .....
}
```

You may, of course, decide to change the function definition to ANSI style. It may be worthwhile examining invocations of the function in question to avoid additional problems.

Promotion (and again)

temp.c:

```
#include <stdio.h>


int
main( void )
{
  int            a = -1;
  unsigned char b = 1;

  if( a < b )
     printf( "%d is less than %u\n", a, b );
  else
     printf( "%d is NOT less than %u\n",
             a, b );
  return( 0 );
}
```

```
% cc temp.c
"temp.c", line 9: warning: semantics of "<"
change in ANSI C; use explicit cast
```

- **With -Xs and -Xt:**

```
% ./a.out
-1 is NOT less than 1
```

- **With -Xa and -Xc:**

```
% ./a.out
-1 is less than 1
```

With K&R semantics, the combination of signed and unsigned types result in an unsigned type (meaning that a negative quatity may lose its sign).

With ANSI, a different *value preserving* approach is applied to promotions meaning that the resultant type may be signed or unsigned depending on the *relative size of the operand types*.

Note that the bit patterns being comared are identical, just the way in which the variable **`b's`** type is promoted changes.

It is also worth noting that if **`b`** was an **`unsigned int`** then -1 is never less than 1 regardless of the compiler semantics in use.

Why?

Varargs / Stdarg

K&R 'C':

```
#include <varargs.h>

err( va_alist )
        va_dcl
{
        va_list  ap;
        char    *fmt;
        int      level;

        va_start( ap );
        level = va_arg( ap, int );
        if( level >= DEBUG_LEVEL ) {
                fmt = va_arg( ap, char * );
                ...
        }
        va_end( ap );
}
```

ANSI 'C':

```
#include <stdarg.h>

err( int level, ... )
{
        va_list  ap;
        char    *fmt;

        va_start( ap, level );
        if( level >= DEBUG_LEVEL ) {
                fmt = va_arg( ap, char * );
                ...
        }
        va_end( ap );
}
```

Variable argument functions will have to be converted from <varargs.h> to <stdarg.h>

The ANSI ‘C’ standard requires at least one “*typed*” argument to be declared in the “*stdarg*” function.

ANSI ‘C’ also *requires* a prototype for a variable number of arguments function at function invocation time.

With -X{s,t,a} you have “Sun extensions” that allow no first argument and the va_start(ap,); macro to be called with no second argument.

We do not recommend using vendor extensions, they make your code non-portable.

Preprocessing

String literals

```
#define string( c ) "c"

main()
{
        printf( "%s\n", string(testing) );
}
```

TABLE 3. String literals

Option	Error / Warning	Output
-Xs	None	"testing"
-Xt	Warning	"testing"
-Xa	Warning	"c"
-Xc	None	"c"

ANSI C does not do token replacement within string literals.

Note the silent one!

- -Xt yields: warning: macro replacement within a string literal
- -Xa yields: warning: no macro replacement within a string literal

The difference is subtle.

The ANSI way:

```
#define string( c )  # c


main( void )
{
        printf( "%s\n", string(testing) );
}
```

Preprocessing

Token pasting

```
#define glue( a, b )   a/**/b

main()
{
        int x1 = 10;

        printf( "%d\n", glue(x,1) );
}
```

TABLE 4. Token pasting

Option	Error / Warning	Output
-Xs	None	10
-Xt	Warning	10
-Xa	Error	N/A
-Xc	Error	N/A

ANSI C treats comments as white space.

- -Xt yields: warning: comment is replaced by "##"

The ANSI way:

```
#define glue( a, b )   a ## b

main()
{
        int x1 = 10;
        printf( "%d\n", glue(x,1) );
}
```

POSIX

IEEE Portable Operating System Interface for Computing Environments

Published by ANSI as IEEE 1003.1-1988
by ISO as ISO/IEC 9945-1: 1990
(IEEE 1003.1a-1990)

Unix System V Release 4.0 (and therefore Solaris) is POSIX 1003.1a conforming.

TABLE 5. POSIX new numbering scheme-

New	Old	Description	Solaris 2.x
1003.1a	same	System Interface	now
1003.1b	1003.4	Real Time Ext.	still a ways off
1003.1c	1003.4a	Threads Interface	early access
1003.1g	1003.12	Network Interface	still a ways off
1003.2	same	Shell & Utilities	now (optional)

POSIX 1003.1a defines a standard way for an application to obtain basic services from the operating system.

POSIX 1003.2 defines a standard way for the shell and it utilities to operate. It is co-bundled with Solaris 2.4 in the SUNWposx2 package.

Derived from a combination of AT&T Unix System V.3 and Berkeley Standard Distribution (BSD) Unix.

The POSIX functions are in the system libraries. Defining the symbol "_POSIX_SOURCE" indicates that all vendor extensions are to be hidden from the standard header files (not unlike __STDC__ in the ANSI 'C' spec).

If you include a header file not specified by the standard all the symbols in that header will be seen.

TABLE 6. Header files (excluding ANSI 'C' headers)

dirent.h	fcntl.h	grp.h	pwd.h	
sys/stat.h	sys/times.h	sys/types.h	sys/utsname.h	sys/wait.h
tar.h	termios.h	unistd.h	utime.h	

System V Release 4 (SVID III)

Solaris is
"System V Interface Definition Release III"
(SVR4) compliant.

It's not
"What did they take out of SunOS?"
but
"What did they put back into SVR4?".

SVID3 is comprised of a number of pieces. Solaris is compliant with the Base System, Basic Utilities, Kernel, Network Services, Terminal Interface Extensions, Advanced Utilities and Software Development Extensions.

(Volumes 1-4)

SVID3 is delivered in 5 volumes, the fifth of which contains extensions to the first four. Solaris is not entirely compliant with the extensions.

X/Open Portability Guide (XPG)

Solaris 2.4 is a conforming implementation of XPG4 (Base Brand).

X/Open is an international consortium of system vendors, ISVs, and users. Adopting existing standards and adapting them to create a consistent environment.

XPG4 incorporates POSIX 1003.1a and1003.1b.

Issue 4 was published in 1992. It did not include a network services portion (not unlike POSIX 1003.1a).

Base brand means that conformance is limited to the base level of conformance in the areas of:

- Window management
- Commands and Utilities
- C Language Component
- Magnetic Media (Source Code Transfer)
- Inter-Process Communication (*SysV IPC*)
- Terminal Interfaces (XSI Curses Interface)

XPG namespace is available via the _XOPEN_SOURCE & _XOPEN_VERSION feature test macros.

Spec 1170

X/Open CAE Specification
consisting of:

- *System Interface Definitions, Issue 4, Version 2*
- *System Interfaces and Headers, Issue 4, Version 2*
- *Commands and Utilities, Issue 4, Version 2*
- *Networking Services, Issue 4*

Collectively refered to as Spec 1170, these specifications comprise one of the major milestones in the COSE (Common Operating System Environment) agreement.

CAE == Common Applications Environment

Spec 1170 revises and builds on XPG4 (adds the missing network component).

Additionally Spec 1170 was derived by analyzing API usage by real applications. As such, many of the API functions that were absent in previous releases of Solaris, re-appear in Solaris 2.5.

Solaris 2.5 is not entirely compliant with the Networking Services portion of the standard. (no XTI support)

Namespace is also controlled by _XOPEN_SOURCE & _XOPEN_VERSION.

Implementation Defined

When a standard calls something implementation defined, what does it mean and where can I find more information?

Sometimes a standard cannot define exactly how an implementation will handle a particular issue (hardware differences are perhaps the most common rationale) and hence will use the term *implementation defined*.

Both the POSIX and the X/Open standards require that vendors supply a conformance document which clearly states how the implementation handles specific situations.

This document is available in the

Software Developer AnswerBook,
Standards Conformance Reference Manual.

An excerpt:

2.7.2. POSIX.1 Symbols

P. Implementations, future versions of this part of ISO/IEC 9945, and other standards may define additional feature test macros.

S. Additional defined feature test macros: _XOPEN_SOURCE, _POSIX_C_SOURCE, __REENTRANT and _KERNEL. Use of these macros is described in the *X/Open Portability Guide Issue 3* and IEEE Std 1003.1b-1993.

Networking Protocols

Solaris 2.x complies with these Internet Engineering Task Force (IETF) Request for Comments (RFC):

TABLE 7. Solaris RFC compliance

RFC	NAME
768	UDP
791	IP
792	ICMP
793	TCP
826	ARP
903	RARP
950	Subnetting Procedure
1058	RIP
1256	ICMP Router Discovery
1331	PPP Multicast
1332	PPP IPCP

Solaris compliance causes some behaviour changes from SunOS 4.x (more on this later).

The IETF has taken over from the “Defence Advance Research Projects Agency” (DARPA) in managing the RFCs that define the way that TCP/IP network protocols work.

The RFCs are FTP’able from ds.internic.net

Porting Tools & Options

- Porting Options
- Solaris Transition Tool
- lint
- cproto
- Editors
- Using cpp, or ld
- Configure
- SparcWorks debugger

Porting Options

1 - Don’t port your code

use Binary Compatibility

2 - Don’t port your code

use Source Compatibility

3 - Port your code

go all the way (see next slide)

You can try the binary compatibility mode to run SunOS 4.x applications on Solaris 2.x, if the BCP is installed.

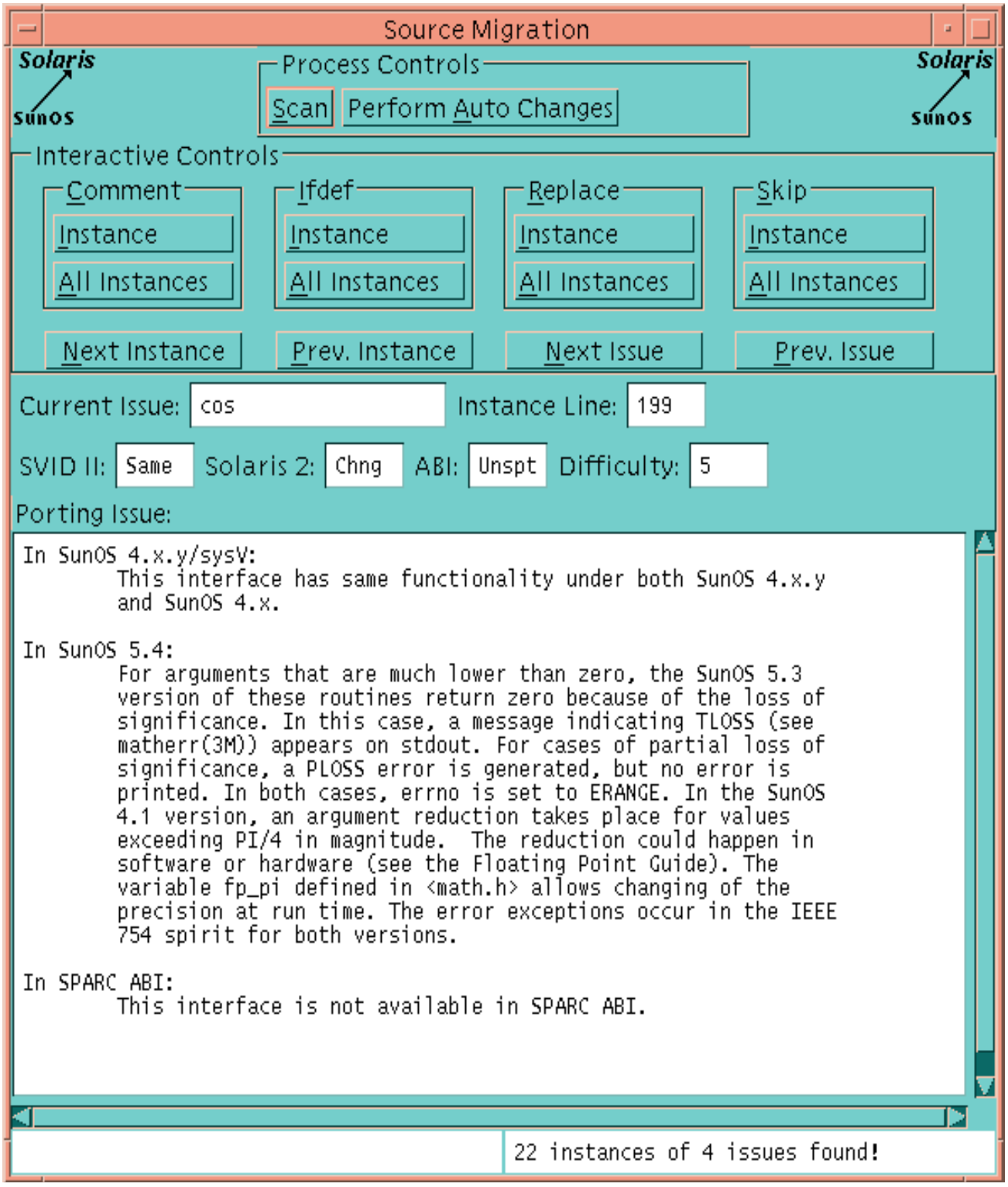
The application must follow these rules:

- It is dynamically linked
  - 2.3 can handle *all* statically or *all* dynamically linked
  - 2.5 can handle mixed (with a kernel variable setting)
- Does not use unpublished system calls
- Does not rely on the name, or format of system admin. files.
- Does not use libkvm, /dev/mem, or /dev/kmem
- Does not use customer supplied device drivers or ioctl()s
- Does not trap directly into the kernel.

You can try to compile your application using the source compatibility libraries and header files under /usr/ucblib and /usr/ucbinclude on Solaris 2.x (use /usr/ucb/cc)

You can NOT mix SysV.4 libraries and UCB libraries or header files.

Solaris Transition Tool.



The Solaris Transition Tool is the replacement for the Pipeline tool.

The biggest improvement over pipeline is that *stt* is able to modify your code.

The single biggest problem with *stt* is that it is not built on a language parser and as such cannot gracefully deal with things like:

```
bcopy(index(string, 'a'), buffer, 5);
```

Stt checks for compliance with the System V.3 API set and the Solaris 2.4 API.

While what is pictured is the *xmstt* (Motif version) the tool is runnable on either Solaris 2.x or SunOS 4.1.x in command line or GUI form (4.1.x only supports the OpenLook GUI style).

Freely available (see Internet Resourses section).

Lint

```
05 main( void )
06 {
07     char c;
08
09     c = getchar();
10     while ( c != EOF ) {
11             (void) printf( "%c", c );
12             c = getchar();
13     }
14     return( 0 );
15 }
```

```
lint junk.c

<No problems reported>
```

```
lint -p -Xc junk.c
(10) warning: suspicious comparison of char
with negative constant: op “!=”
```

There are more lint libraries in Solaris 2.x than in SunOS 4.x, but still none for xview, or olit.

The biggest problem with *lint* is that lint libraries are frequently out of sync with reality.

lint -p -Xc -Idir file.c file.c file.c...

For portable code use the -p and / or -Xc option(s).

See section 8 of the SPARCompiler ‘C’ Programmer’s Guide for more information on lint and the messages from lint.

There are a number of ways to do prototype checking:

- Using “lint” and looking for *implicit function declaration* errors.
- Using SunPro “cc” with the ‘-v’ option and looking for *implicit function declaration* errors.
- Using a ‘C++’ compiler and looking for *undefined function* errors.
- Using gcc 2.x with the ‘-Wstrict-prototypes’ option to do strict prototype checking.

This will help find argument promotion problems.

cproto

```
main( argc, argv )
  int argc;
  char *argv[];
{
  char c;

  c = getchar();
  while( c!= EOF ) {
      printf( "%c",c );
      c = getchar();
  }
  return( 0 );
}
```

Running cproto

```
% cproto -a -e -p *.c >prototypes.h
extern int main( int argc, char *argv[] );
```

Looking at the file again.

```
#include "prototypes.h"      /* you must add this line */

int
main( int argc, char *argv[] )
{
  char c;

  c = getchar();
  while( c!= EOF ) {
      printf( "%c",c );
      c = getchar();
  }
  return( 0 );
}
```

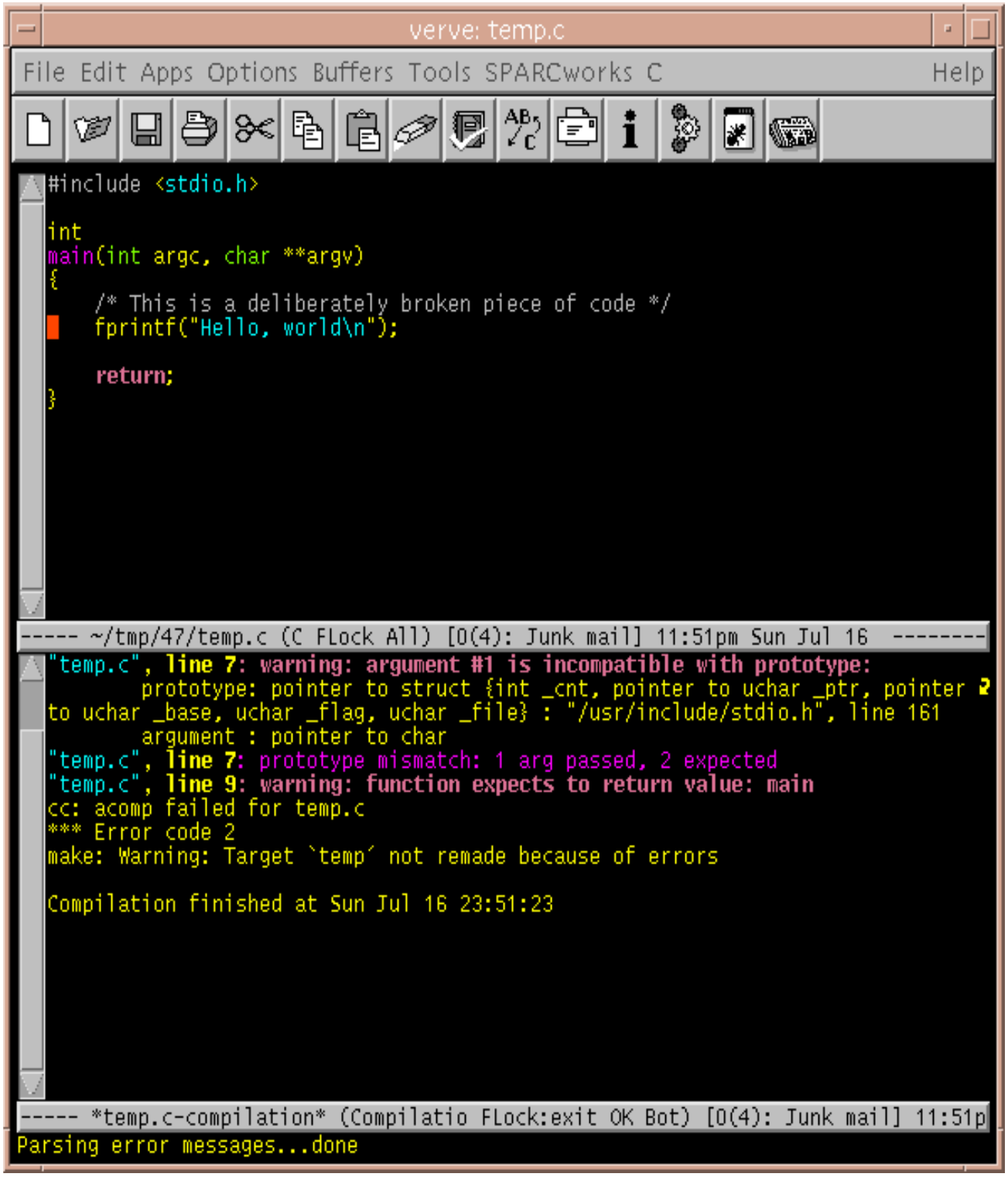
Cproto reads K&R style code, and converts the function declarations to ANSI style, and produces prototypes on stdout.

```
usage: cproto [option] [file]
Options:
 -a      Convert function definitions to ANSI style
 -c      Omit comments in generated prototypes
 -e      Output “extern” keyword before global declarations
 -f n    Set function prototype style (0 to 4)
 -p      Disable formal parameter promotion
 -s      Output static declarations
 -t      Convert function definitions to traditional style
 -v      Output variable declarations
 -m name Set name of prototype macro
 -d      Omit prototype macro definition
 -P fmt  Set prototype format template “int main (a, b)”
 -F fmt  Set function definition format template “int main (a, b)”
 -C fmt  Set format for function definition with parameter comments
 -V      Print version information
Set C preprocessor options
 -D name[=value]
 -U name
 -I directory
```

Cproto is free software and on the OpCom ftp server.

Another option is GNU *protoize*, which comes with GNU *gcc* or stand-alone.

Editors


verve: temp.c
File Edit Apps Options Buffers Tools SPARCworks C Help
#include <stdio.h>
int
main(int argc, char **argv)
{
/* This is a deliberately broken piece of code */
fprintf("Hello, world\n");
return;
}
----- ~/tmp/47/temp.c (C FLock All) [0(4): Junk mail] 11:51pm Sun Jul 16 --------
"temp.c", line 7: warning: argument #1 is incompatible with prototype:
prototype: pointer to struct {int _cnt, pointer to uchar _ptr, pointer to uchar _base, uchar _flag, uchar _file} : "/usr/include/stdio.h", line 161
argument : pointer to char
"temp.c", line 7: prototype mismatch: 1 arg passed, 2 expected
"temp.c", line 9: warning: function expects to return value: main
cc: acomp failed for temp.c
*** Error code 2
make: Warning: Target `temp' not remade because of errors
Compilation finished at Sun Jul 16 23:51:23
----- *temp.c-compilation* (Compilatio FLock:exit OK Bot) [0(4): Junk mail] 11:51p
Parsing error messages...done

An integrated compile/edit cycle can increase programmer performance.

To this end, most OpCom engineers use some form of *emacs* to help in the porting effort.

What editor you choose is not important. The features that you should look for in a programmer's editor are:

- multiple file
- integrated compile/edit
- macro language

XEmacs can run "*make -k*" from the command buffer, then any errors are placed in the compilation buffer.

- When the make command exits, you can use next-error, prev-error macros to visit each error message.
- When you visit an error message, the file where the error occurred is loaded into an edit buffer and the cursor is placed on the line at fault.
- Now just fix the problem, and press next-error.

Other editors that can cycle through errors include *sam, crisp*, *brief* and *codecenter.*

emacs macros.

```
(defvar ifdef-text “def __SVR4”
    “Variable containing the cpp label to be used in the
     construction of the ifdef block”)

(defun insert-ifdef-region ()
    “Function to duplicate the current region inserting
     an #ifdef/#else/#endif block around the duplicated
     lines appropriately”
    (interactive)
    (let ((start-position (make-marker))
            (end-position (make-marker)))
        (set-marker start-position (mark))
        (set-marker end-position (point))
        (kill-region start-position end-position)
        (insert (concat “\n#if” ifdef-text “\n”))
        (yank)
        (insert “#else\n”)
        (yank)
        (insert “#endif\n”)))
```

Another way, for non-*emacs* users.

```
#!/bin/sh
#  Shell script to duplicate the text on stdin
#  inserting an #ifdef/#else/#endif block
#  around the duplicated lines appropriately

cat  >/tmp/$$
echo  "#ifdef ${IFDEF_LABEL:-__SVR4}"
cat   /tmp/$$
echo  "#else"
cat   /tmp/$$
echo  "#endif"
rm    /tmp/$$
```

This *emacs* macro will place
#ifdef __SVR4
around a block of selected code, and then place
#else #endif
around a copy of the same block.

You can then edit the copied block to make the Solaris changes.

e.g.

```
#ifdef __SVR4
        fds = sysconf( _SC_OPEN_MAX );
#else
        fds = getdtablesize();
#endif
```

Using cpp, or ld

file.c:

```
#if defined( sun ) && !defined( __SVR4 )
#   include "sunos.h"
#endif
...
```

sunos.h:

```
#define memcpy( d, s, l )  bcopy( s, d, l )
#define strchr( s, c )     index( s, c )
etc...
```

```
cc file.c sunos.c
```

sunos.c:

```
memcpy( char *d, char const *s, int const l )
{
        return( bcopy(s, d, l) );
}

strchr( char const *s, char const c )
{
        return( index(s, c) );
}
```

If you have to support both SunOS and Solaris versions of the application it is a good idea to do so from one source tree.

By including the sunos.h file in your source, you can use *cpp* to convert the code as needed.

The sunos.c file is the same idea, but you supply the missing functions and only link if you are on SunOS.

It is best to use this method to support the new interfaces on the old platform so that when support for SunOS can be dropped it is simply a matter of removing the “sunos.h” file.

Configure

sysconf.h:

```
#define HAVE_STRING_H
#define HAVE_VPRINTF
#define HAVE_WAIT3
#define HAVE_VFORK
...
```

func.c:

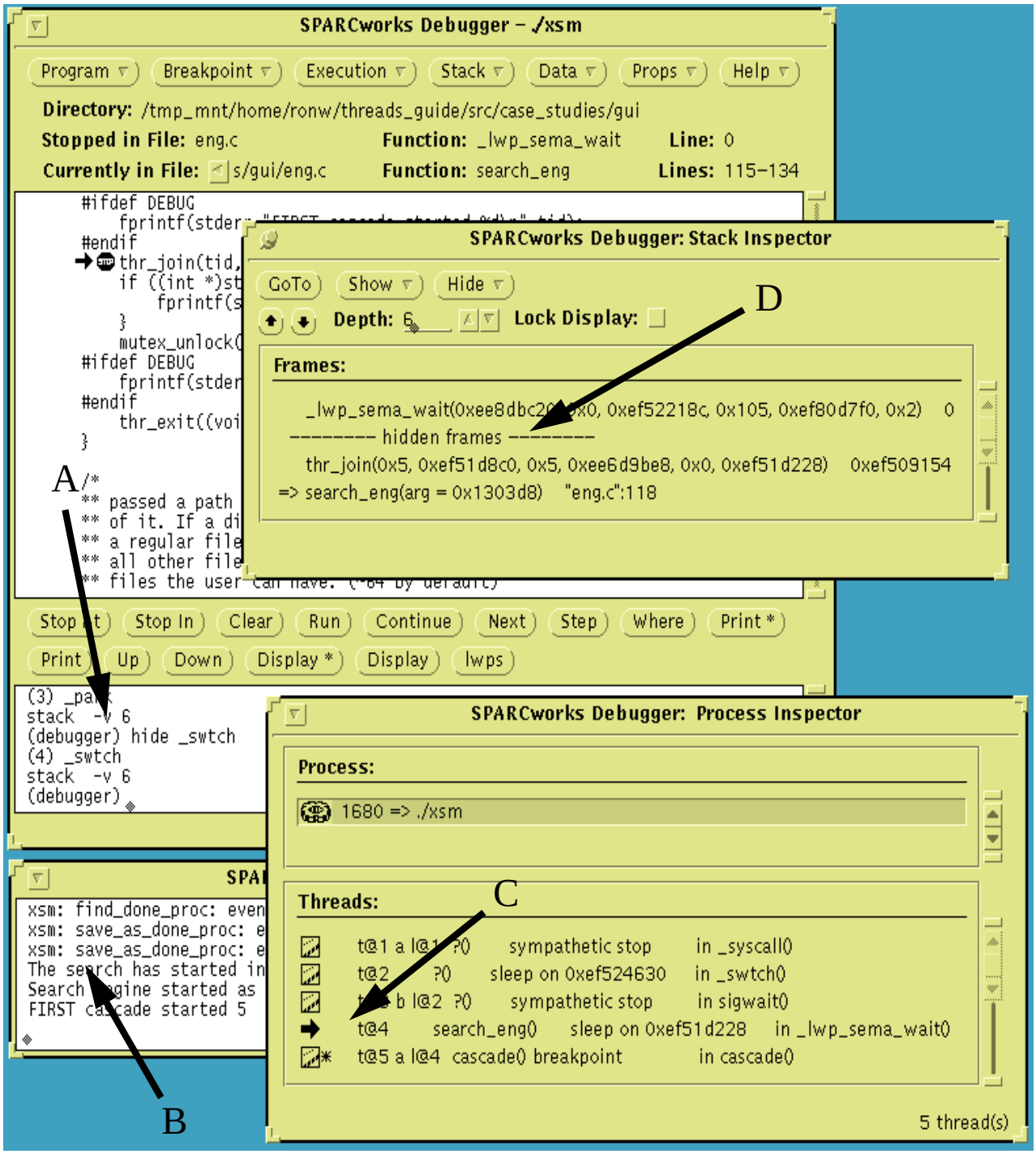
```
#include "sysconf.h"

#if   defined( HAVE_VPRINTF )
       vfprintf( stderr, fmt, args );

#elif defined( HAVE_DOPRNT )
       _doprnt( fmt, args, stderr );

#else
#      error need HAVE_VPRINTF or HAVE_DOPRNT
#endif
...
```

Another method is to build a **sysconf.h** file specifying what interfaces are supported on the platform you are porting to, and using macros to select the right interface.

It is possible to write a shell script to “probe” the system for this compliance information and generate the **sysconf.h** header file.

Larry Wall’s *MetaConfig* and the Free Software Foundation’s *AutoConfig* systems will write this shell script for you (called **Configure**) when given a list of interfaces you are interested in.

SparcWorks debugger



dbxtool => debugger

dbx

The debugger and dbx are now unbundled in the SparcWorks package.

- A ksh interface to dbx. Can write ksh scripts to examine data.
- B output window. (program output no longer mixed with dbx)
- C Process Inspector. Lists the threads in a multi-threaded app.
- D Stack Inspector. Shows traceback, can hides frames, etc...

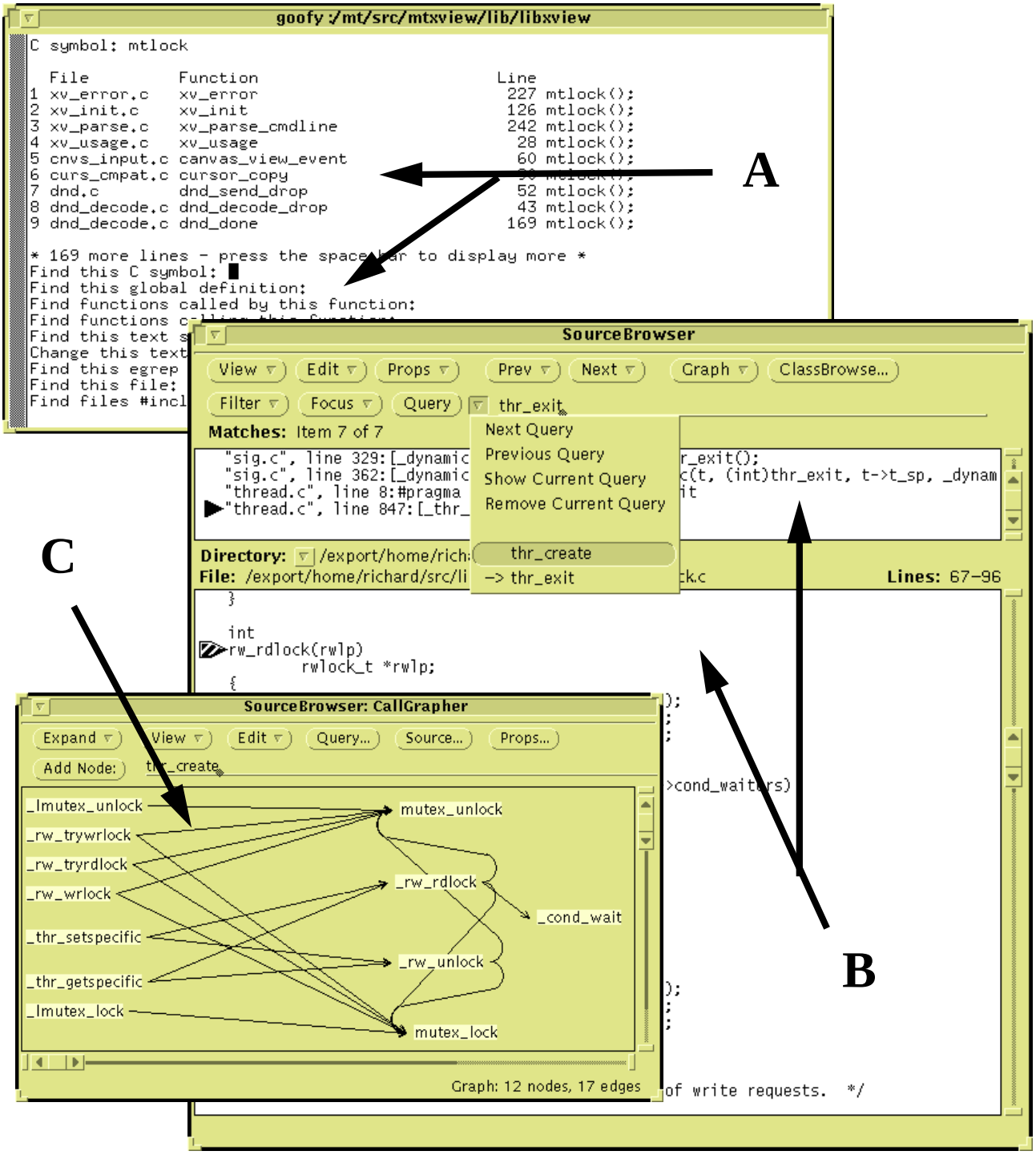
dbx -C will do runtime checking of memory usage (like *Purify*). The “fix” command will re-compile and re-link a fixed function into the debugged process. You can now continue to debug without reloading and losing state information.

New commands include, frame, hide, events (modify), etc.

New commands for threaded programs: lwp, lwps, thread and threads.

There are more thread-smart tools coming (locklint, thead analyzer, etc.).

Other tools


goofy :/mt/src/mtxview/lib/libxview
C symbol: mtlock
File Function Line
1 xv_error.c xv_error 227 mtlock();
2 xv_init.c xv_init 126 mtlock();
3 xv_parse.c xv_parse_cmdline 242 mtlock();
4 xv_usage.c xv_usage 28 mtlock();
5 cnvs_input.c canvas_view_event 60 mtlock();
6 curs_cmpat.c cursor_copy 52 mtlock();
7 dnd.c dnd_send_drop 43 mtlock();
8 dnd_decode.c dnd_decode_drop 169 mtlock();
9 dnd_decode.c dnd_done
* 169 more lines - press the space bar to display more *
Find this C symbol:
Find this global definition:
Find functions called by this function:
A
SourceBrowser
View Edit Props Prev Next Graph ClassBrowse...
Filter Focus Query thr_exit
Matches: Item 7 of 7
Next Query
Previous Query
Show Current Query
Remove Current Query
thr_create
-> thr_exit
Directory: /export/home/rich
File: /export/home/richard/src/li
Lines: 67-96
int
rw_rdlock(rwlp)
rwlock_t *rwlp;
C
SourceBrowser: CallGrapher
Expand View Edit Query... Source... Props...
Add Node: thr_create
_lmutex_unlock
_rw_trywrlock
_rw_tryrdlock
_rw_wrlock
_thr_setspecific
_thr_getspecific
_lmutex_lock
mutex_unlock
_rw_rdlock
_cond_wait
_rw_unlock
mutex_lock
Graph: 12 nodes, 17 edges
B

- A: *Cscope*, curses based tool to browse source code. You can search for symbol definition and use, strings, include files, and egrep expressions. There is also a *cscope* mode for *emacs*.

- B: *SourceBrowser*, an X window based tool to browse source code. Features include, query history, symbol filtering, narrowed focus, C++ class browsing, and the ability to call up the *Debugger*, or *CallGrapher.*

- C: *CallGrapher*, an X window based tool to provide a graphical representation of the code flow. You can query any graphed symbol from this tool into the *SourceBrowser.*

Porting Issues

- Library call changes
- Networking
- Signals
- Ioctl
- ouch!
- Windowing Systems
- Building Shared Objects

Library call changes

Table B-1 System Calls Reference Table (Continued)

SunOS 4.1 System Call	SunOS 5.2	Notes	ABI	SVID	SVR4	BSD
kill(2V)	C	In the SunOS 4.1 release, if a signal is sent to a group of processes (as with, if *pid* is 0 or negative), and if the process sending the signal is a member of that group, the signal is not sent to the sending process as well. In the SunOS 5.2 release, or the ABI, SVID, or SVR4, the signal is sent to the sending process as well. In SunOS 4.1, the *pid* argument is of type int, while in the SunOS 5.2 release, or the ABI, SVID, or SVR4, the *pid* argument is of type pid_t. Also, the SunOS 5.2 release, or the ABI, SVID, or SVR4 version includes <sys/types.h> while the SunOS 4.1 version does not.	C	C	C	N
kill(2V) -- SysV	S		S	S	S	N
killpg(2)	A	The kill(2) system call provides similar functionality. Replace killpg(*pgrp*, *sig*) with kill(-*pgrp*, *sig*).	A	A	A	S
link(2V) -- SysV	C	In the SunOS 5.2 release, or the ABI, SVID, or SVR4 version of link(), if the last component of the first argument is a symbolic link, it will not be followed and a hard link will be made to the symbolic link.	C	C	C	N
listen(2)	S	Now listen(3N).	N	N	S	N
lseek(2V) – SysV	S		S	S	S	N
lstat(2V) – SysV	S		S	S	S	N
mctl(2)	A	The memcntl(2) system call provides similar functionality.	A	A	A	S
mincore(2)	C	In the SunOS 4.1 release, argument *len* is of type int, while in SVR4 and SunOS 5.2, argument *len* is of type size_t which is defined to be unsigned int. The SunOS 5.2 release version also requires inclusion of <unistd.h>.	N	N	C	N

Solaris 2.x System Administrator's AnswerBook,
Solaris 1.x to Solaris 2.x Transition guide:

Appendix B: System Calls Reference Table
and
Appendix C: Library Routines Reference Table.

Symbol codes	
N	Not Available
A	Alternative interface
C	Changed
S	Same
-SysV	in the SunOS 4.1.x System V option.

BSD column - code for how symbol appears in the Solaris 2.x "Source Compatibility Libraries"

System call changes

SunOS	Solaris
getdtablesize()	sysconf(_SC_OPEN_MAX)
gethostid()	sysinfo(SI_HW_SERIAL)
gethostname()	sysinfo(SI_HOSTNAME) uname()
getpagesize()	sysconf(_SC_PAGESIZE)
killpg()	kill(-pid)
mctl()	memctl()
[f]statfs()	[f]statvfs()
setreuid()	setuid() / seteuid()
wait3() / wait4() union wait	wait() / waitpid() WIFEXITED(status), etc.

These are some of the more common system call changes that will be encountered.

Some code examples follow...

System call changes

```
#ifndef HAVE_SYSCONF
        open_max = getdtablesize();
#else
        open_max = sysconf( _SC_OPEN_MAX );
#endif



#ifdef HAVE_KILLPG
        killpg( p_group_id, SIGTERM );
#else
        kill( - p_group_id, SIGTERM );
#endif



        waitpid( -1, &status, WNOHANG );
#ifdef HAVE_UNION_WAIT
        if( status.w_stopval != WSTOPPED )
                printf( "exit status %d",
                        status.w_retcode );
#else
        if( WIFEXITED(status) )
                printf( "exit status %d",
                        WEXITSTATUS(status) );
#endif
```

Most configuration information calls have become either sysconf() or sysinfo() calls.

N.B. getdtablesize() returns in Solaris 2.5 (spec1170).

Use the macros instead of the wait bitfield union. The macros hide the implementation details in a more portable manner.

Libc changes

	SunOS	Solaris
	bcopy()	memmove()
	bzero()	memset(0)
	bcmp()	memcmp()
	on_exit()	atexit()
	dbm functions become dbm_*	
	e.g. fetch()	dbm_fetch(DBM *)
‡	getwd()	getcwd()
	index() / rindex()	strchr() / strrchr()
‡	mkstemp()	mktemp() + open(O_EXCL)
	random()	rand()
‡	re_comp()	regcmp()
‡	re_exec()	regex()
‡	strptime()	getdate()
‡	ualarm() / usleep()	setitimer() + sigaction()
	termcap	terminfo
	struct sgttyb / tchars / lchars	struct termios
	gtty() / stty()	tcgetattr() / tcsetattr()
	struct direct <sys/dir.h>	struct dirent <dirent.h>

‡ returns due to spec 1170

These are some of the more common library changes that will be encountered.

`atexit()` - registered functions do not take arguments.

`regcmp()` - regular expression language has a richer set of meta-characters.

N.B `re_comp()` and friends return in Solaris 2.5 but are slated for removal in the next revision to Spec 1170.

`getdate()` - must set `DATEMSK` environment variable to point to a template file.

N.B. `strptime()` returns in 2.4.

Some code examples follow...

Libc changes

```
#ifdef HAVE_GETWD
    char cwd[MAXPATHLEN];
    getwd( cwd );
#else
    int   size = 256;
    char *cwd  = malloc( size );

    while(  getcwd(cwd, size) == NULL
          && errno == ERANGE )
            cwd = realloc( cwd, size += 256 );
#endif



#ifdef HAVE_MKSTEMP
    strcpy( tmpl, "/tmp/fooXXXXXX" );
    fd = mkstemp( tmpl );
#else
    mode = O_RDWR | O_CREAT | O_EXCL;
    do {
        strcpy( tmpl, "/tmp/fooXXXXXX" );
        mktemp( tmpl );
    } while( (fd = open(tmpl, mode)) < 0 );
#endif
```

Error checking on `malloc(), realloc()` and `getcwd()` left out for brevity.

N.B. `getwd()` returns in Solaris 2.5.

Be aware that `mktemp`() will only give 26 unique file names per root template.

N.B. `mkstemp()` returns in 2.5.

Libc changes

```
#if defined( HAVE_DIRECT )

    #include <sys/dir.h>

    printf( "name (%.*s) is %d chars long\n",
            dir->d_namlen,
            dir->d_name,
            dir->d_namlen );


#elif defined( HAVE_DIRENT )

    #include <dirent.h>
    #include <string.h>

    printf( "name (%s) is %d chars long\n",
            dir->d_name,
            strlen(dir->d_name) );


#else
#   error need HAVE_DIRECT or HAVE_DIRENT
#endif
```

POSIX.1 `struct dirent` does not have the `d_namlen` member **but** the `d_name` member is guaranteed to be `NULL` terminated.

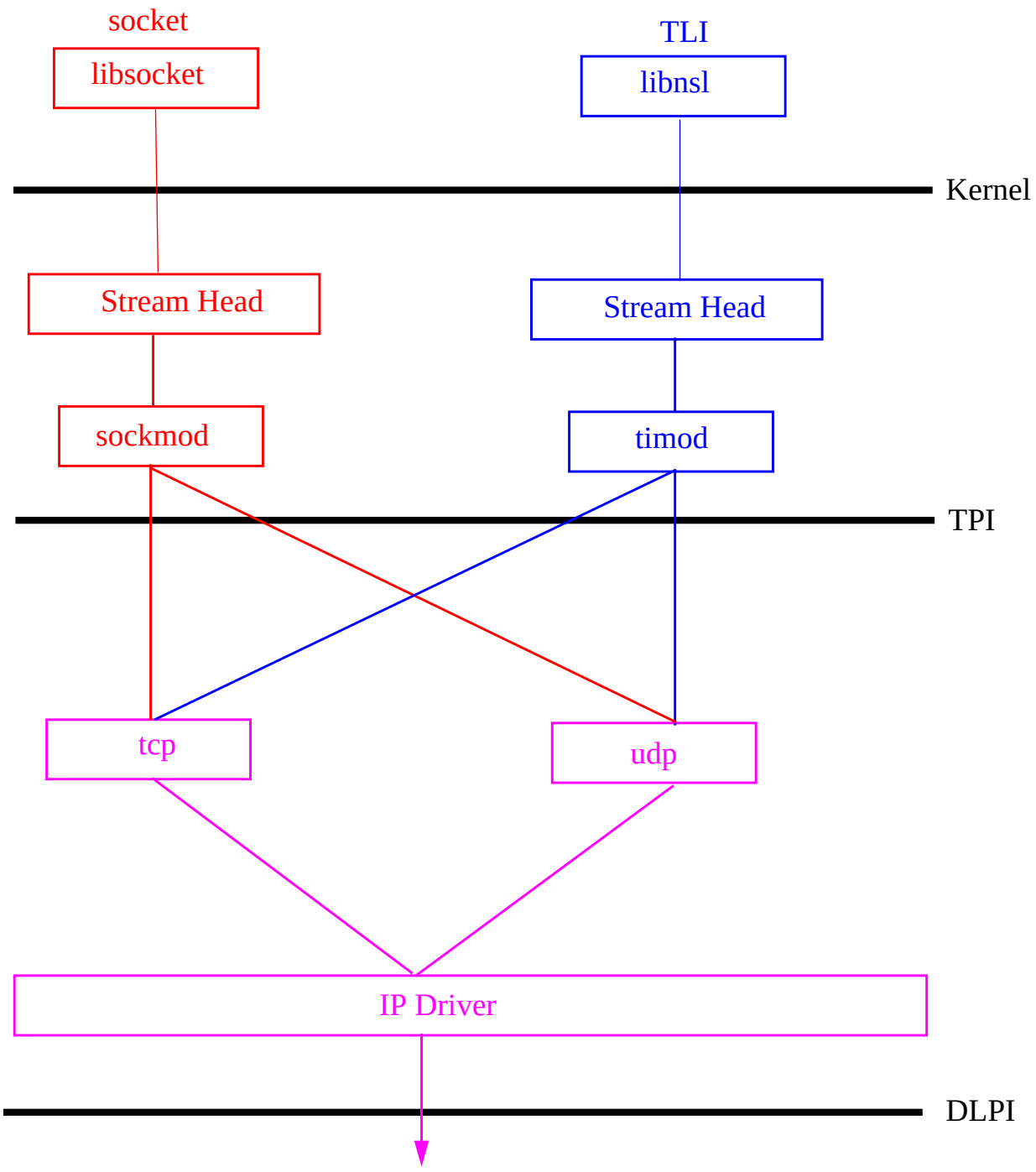
Libc changes

```
#if defined( HAVE_UALARM )

    ualarm( 100 );

#elif defined( HAVE_ITIMER )

    sigset_t         mask;
    struct itimerval tval;

    /* set alarm time */
    tval.it_value.tv_sec  = 0;
    tval.it_value.tv_usec = 100;
    timerclear( &tval.it_interval );

    /* make sure SIGALRM isn't blocked */
    sigemptyset( &mask );
    sigaddset( &mask, SIGALRM );
    sigprocmask( SIG_UNBLOCK, NULL, &mask );

    setitimer( ITIMER_REAL, &tval, NULL );

#else
#   error  need HAVE_UALARM or HAVE_ITIMER
#endif
```

Why have things gotten so complicated?

To make matters still more confusing, `ualarm()` and `usleep()` have returned in Solaris 2.5.

Networking


socket
libsocket
TLI
libnsl
Kernel
Stream Head
Stream Head
sockmod
timod
TPI
tcp
udp
IP Driver
DLPI

Sockets are not defined as part of the ABI, but are a Sun value add (also in the POSIX 1003.1g draft and Spec. 1170).

Sockets are no longer a kernel-level interface (system calls), but instead sit on top of networking STREAMS modules.

libsocket, libnsl: Socket functions are no longer in libc

There are performance issues when using the Socket or TLI connection routines [`sock(), accept(), connect(), t_open(), t_accept(), t_connect()`] they must set up or tear down STREAM stacks.

You should avoid opening and closing, and then re-opening connections. The cost of STREAM setup, and tear down is too high.[1]

NIT (Network Interface Tap) is no longer needed or supported, the Ethernet drivers are real STREAMS drivers. (see *man -s7 le*, *ie*; for more information)

TPI = transport provider interface, protocol independent.

DLPI = data link provider interface, hardware independent.

1. The data performance of Solaris 2 sockets / TLI is higher then SunOS 4.1.x

Sockets

```
s = socket( AF_INET, SOCK_STREAM, 0 );
if( s < 0 ) {
    perror( "open socket" );
    exit( -1 );
}
...

ret = bind( s, &addr, sizeof(addr) );
if( ret < 0 ) {
    close( s );
    perror( "bind" );
    exit( -1 );
}
...

while( 1 ) {
    msgsock = accept( s, 0, 0 );
    if( msgsock < 0 ) {
        if( errno == EINTR )
            continue;
        perror( "accept" );
        exit( -1 );
    }
    ...
    close( msgsock );
}
```

Since the Socket API is no longer a set of system calls but is instead a set of library routines over a stack of Streams modules, those routines are no longer restarted when a signal is delivered (i.e. **not** signal safe).

Don’t trust the man pages all the Socket API functions can be interrupted (set `errno` to **`EINTR`**).

May be fixed in a future release.

SO_REUSEADDR

```
s = socket( AF_INET, SOCK_STREAM, 0 );
if( s < 0 ) {
    perror( "open socket" );
    exit( -1 );
}

...

setsockopt( s, .., SO_REUSEADDR, &on, .. );

ret = bind( s, &addr, sizeof(addr) )
if( ret < 0 ) {
    close( s );
    perror( "bind" );
    exit( -1 );
}

setsockopt( s, .., SO_REUSEADDR, &off, .. );

...
```

Solaris is now compliant with the TCP RFCs.

One consequence of this is that after an **active** *close* of the endpoint TCP will wait 2*MSL (maximum segment lifetime) before allowing the socket pair to be reused [called TIMED_WAIT][1].

An **active** *close* occurs on the side that calls `close()`, either directly or indirectly [via `exit()` or interrupt].

A **passive** *close* occurs on the side that receives the FINish packet from the side that did the **active** *close*.

Putting it all together:

If a service provider (daemon) dies due to an interrupt then that service provider can not be restarted for 2*MSL [default **four minutes**, configurable using ndd(1m)].

One way around this is to use a `setsockopt(SO_REUSEADDR)` to allow an endpoint (local port number) in TIMED_WAIT to be reused.

This circumvents protection against multiple instances of the same daemon being started. Could use a xxx.pid file or try to `connect()` to the server port to put this protection back in.

1. see "TCP/IP Illustrated, Volume 1" by W. Richard Stevens, pgs. 242-245

Asynchronous Connect

SunOS 4.x:

```
s = socket( AF_INET, SOCK_STREAM, 0 );
if( s < 0 ) {
    perror( "open socket" );
    exit( -1 );
}

...

signal( SIGIO, connected );

fcntl( s, F_SETOWN, getpid() );

flags = fcntl( s, F_GETFL, 0 );
flags |= O_ASYNC | O_NONBLOCK;
fcntl( s, F_SETFL, flags );

ret = connect( s, &addr, sizeof(addr) );

...
```

Under SunOS 4.x this code would cause `connect()` to return -1 with `errno` set to `EINPROGRESS`. When the connection finishes a `SIGIO` signal will be delivered and `connected()` [the signal handler] will be called.

This **no longer works** under Solaris 2.x, `connect()` returns -1 with `errno` set to `EINPROGRSS` but the signal is never delivered.

A `select()` or `poll()` call can be used (by selecting for writing) to determine when the connection is finished.

Setsockopt()

SunOS 4.x:

```
setsockopt(s, SOL_SOCKET, SO_DEBUG, 0, ...);
```

Solaris 2.x:

```
int off = 0;
setsockopt(s, SOL_SOCKET, SO_DEBUG, &off, ...);
```

`Setsockopt()` takes a pointer as its fourth argument. The SunOS 4.x system call was nice to you and took a NULL pointer as zero, Solaris **no longer does**.

TI-RPC

TABLE 8. Differences Between TI-RPC and RPC

	Solaris 2.x TI-RPC	SUN OS 4.1.x RPC
Default transport	TLI	Socket interface
Address binding	rpcbind (in universal format)	portmap (port numbers)
Transport Information	Local file: /etc/netconfig. Any transport in this file, accessible.	ONLY TCP / UDP transports supported
Host names	/etc/nsswitch.conf	name service (/etc/hosts & +)
File desciptor	TLI Endpoints	Socket Endpoints
rpcgen	Multi-args, pass-by-value, ANSI C	Multi-args, call-by-value.
Libraries	libnsl needed at link time	all functions in libc
loopback	rpcbind uses secure loopback	Not used

TI-RPC, A new way to deal with RPCs and different transports.

The old interfaces are still available (rpcgen -b). But they only work with TCP / UDP.

#define PORTMAP before #include of <rcp/rpc.h>
add -L/usr/ucblib -lrpcsoc to Makefile

The new transport are TLI based.
TLI using the idea of “opaque addressing”

No more struct sockaddr, use struct netbuf.

Re-running <file>.x through rpcgen on Solaris will give you TI-RPC and ANSI code.

See:

SunOS 5.x Network Interfaces Programmer’s Guide in the Answerbook

or *ONC+ RPC Application Toolkit 2.0.* P/N 801-4207-10

also

the Public Domain source for TI-RPC is on the FTP server.

Signals

SunOS 4.x:

```
#include <signal.h>

main()
{
    void (*old)();

    old = signal( SIGALRM, handler );
}
```

Solaris:

```
#include <signal.h>

main()
{
    struct sigaction act, old;

    sigemptyset( &act.sa_mask );
    act.sa_flags = SA_RESTART;/*default BSD*/
    act.sa_handler = handler;
    sigaction( SIGALRM, &act, &old );
}
```

signal() & sigaction()

The `signal(2)` interface provides System V.3 semantics. The signal handler is reset to `SIG_DFL` before the handler is called, and is **NOT** blocked while in the handler. A second signal of the same type will most likely call exit().

`Sigaction()`, the POSIX.1 interface gives you more control over how the signal handler will behave, and can provided the BSD semantics.
If the handler is set using `sigaction()`, or `sigset()` the signal is automatically blocked while the signal handler is executing, and the handler is preserved.

If the signal handler is exited via `longjmp()` then the signal must be unblocked by the user with `sigprocmask()` first.

If system calls are to be restarted after the return from the signal handler then `sa_flags` must contain `SA_RESTART.`

`SA_RESTART` is not part of POSIX.1, and there is **no** definitive list of which system calls are restartable.

N.B. Solaris 2.5 adds bsd_signal().

All signal code needs re-examination to use the POSIX interface.

Signals

BSD/SunOS	Sys V.3	POSIX.1
int handler()	int handler()	void handler()
signal()	sigset()	sigaction()
signal()	sigignore()	sigaction()
sigvec()	*(none)*	sigaction()
sigblock()	sighold()	sigprocmask()
sigsetmask()	sigrelse()	sigprocmask()
sigpause()	sigpause()	sigsuspend()
(none)	*(none)*	sigpending()
SIGIO		SIGPOLL

More signal interface changes.

ioctl

```
#include <unistd.h>
#include <sys/types.h>
#include <stropts.h>


ioc(int fd, int request, void *arg, int len)
{
        struct strioctl     str;

        if( isastream(fd) ) {
                str.ic_cmd    = request;
                str.ic_dp     = arg;
                str.ic_len    = len;
                str.ic_timout = 0;
                return(ioctl(fd, I_STR, &str));
        }
        return(ioctl(fd, request, arg));
}




#define IOC( fd, req, arg ) \
         ioc( fd, req, arg, sizeof(*arg) )
```

Serial tty interfaces now use transparent `ioctl()`s, but some of the `ioctl()`s are not supported by the stream head using the transparent interface and must be submitted as an *I_STR*.

e.g. TIOCMGET

An `ioctl( fd, TIOCMGET, &flags )` will return success but never set the *flags* argument to the modem control signals. The data is returned by the device driver but thrown away by the stream head.

ouch!

malloc() / free()
realloc()

strftime()

getrusage()/wait3()

getpwent()

select()

Freeing non-`malloc()'d` space will dump core, but the SIGSEGV will occur on a subsequent call to `malloc()` in `realfree()`.

Passing a pointer to `realloc()` that has already been `free()'d` will cause a SIGSEGV (core dump).

The format string for `strftime()` no longer accepts certain format specifiers (%k, %l for example).

NB. back in 2.4

The information returned by `getrusage()` is significantly different. See man proc(4); SunOpsis Volume 1, Number 3; and the Solaris Porting FAQ.

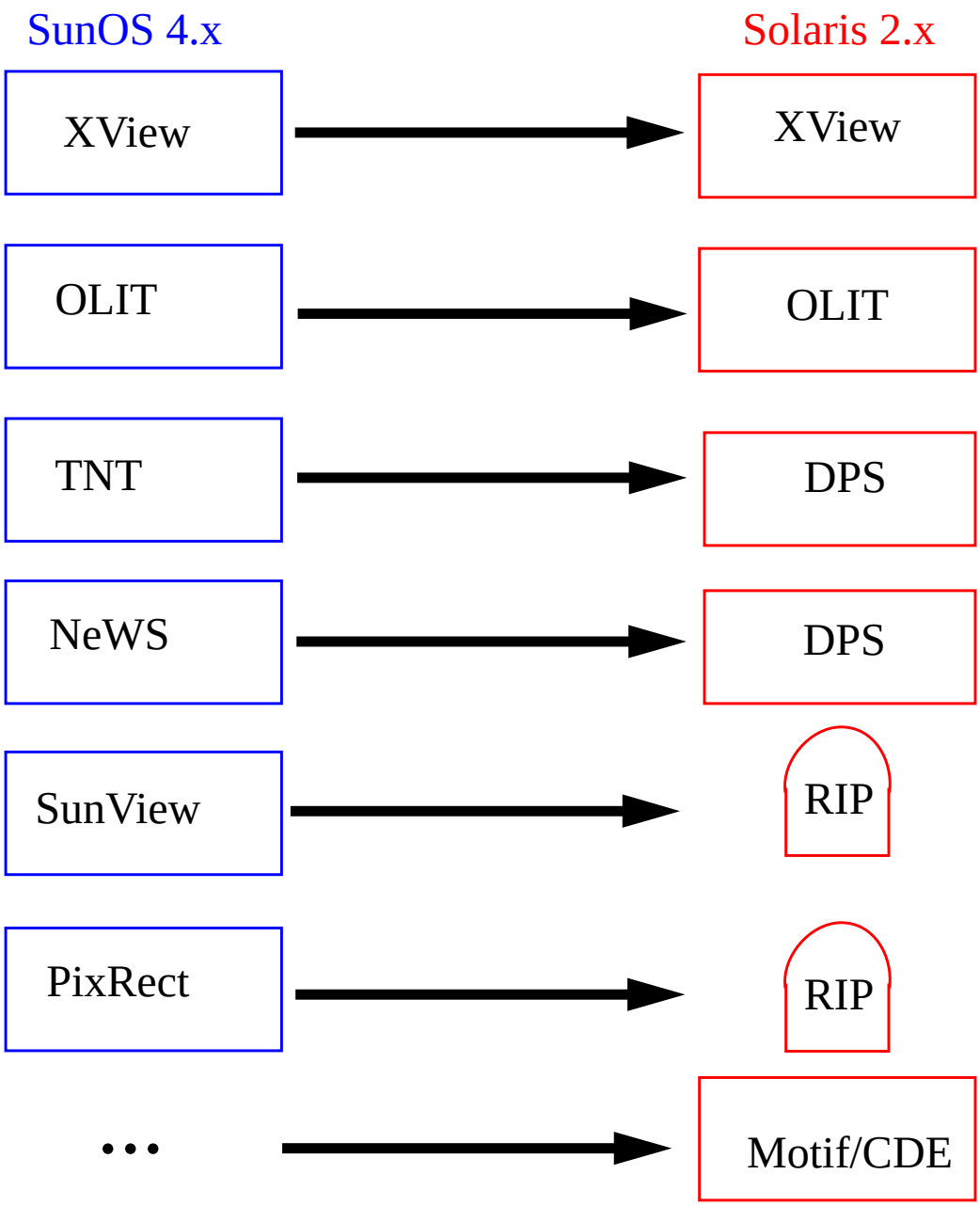
Solaris 2.5 only provides CPU time usage with the `RUSAGE_CHILDREN` argument and as such does not fully support the BSD `rusage()` semantics.

With the /etc/shadow file, when you call `getpwent()`, the structure is returned with the password from /etc/passwd, which is not the real passwd. See getspent(3c).

Also the information returned may be different depending on which name service provider is set in `nsswitch.conf`.

`select()` is implemented using `poll()`, therefore the timeout value only has millisecond resolution.

Windowing Systems



XView 3.x to XView 3.0.x the same.

OLIT the same.

TNT is GONE

NeWS is gone, PostScript support is now available through the DPS extension in the Solaris 2.3 and up X servers.

SunView headers and runtime libraries disappeared in 2.2.
SunView will NOT run under 2.2 or above. Port to XView 3.x.
PixRect libraries are gone, Need to use X11 drawing primitives.

The birth of Motif on Sun.
Motif run-time libraries are in 2.4
CDE bundled with 2.5

devguide 3.x porting.

cproto -a -e *.c
rerun *gxv* or *golit* on *.{GP} files.
This will create ANSI ‘C’ files.

gmf - Motif look and feel from devguide.

Building Shared Objects

SunOS 4.1.x build:

```
%cc -pic -c *.c
%ld -o libname.so.M#.m# -assert pure-text *.o
```

also needed to build libname.sa for init code and data segment.

Solaris 2.x build:

```
%cc -c -K PIC *.c
%ld -G -z text -o libname.so.M# *.o
```

-h libname.so.M# to include version info, No more need for the *.sa

WHY?

Your applications can benefit from using shared objects.

The code set of any application is reduced, and the resources needed to store and execute the shared common functions among applications is reduced.

Bug fixes to common functions can be made with out re-compiling the application, and binding of local functions of the same name can be used to “override” a common function in a application with special needs.

Since the a.out format was replaced with the “Executable and Linking Format” (ELF) the `‘libxx.sa’` file to hold the shared libraries data segment is no longer needed.

Shared objects are a good thing, unless the application is very short lived (where the load time is greater than the run time).

LD_LIBRARY_PATH, LD_RUN_PATH, and the -R compiler flag will all come into play.

Further Reading

- “Advanced Programming in the UNIX Environment”, W. Richard Stevens. Addison-Wesley Professional Computing Series. 1992.“
- “TCP/IP Illustrated, Volume 1”, W. Richard Stevens. Addison-Wesley Professional Computing Series. 1994.“
- “UNIX Network Programming”, W. Richard Stevens. Prentice Hall Software Series. 1990.
- “The POSIX.1 Standard A Programmer’s Guide”, Fred Zlotnick. The Benjamin/ Cummings Publishing Company ISBN 0-8053-9605-5
- “POSIX Programmers’s Guide”, Donald Lewine. O’Reilly & Associates, Inc. 1991
- “Solaris Porting Guide”, SunSoft ISV Engineering. SunSoft Press (PTR PH) A Prentice Hall Title ISBN 0-13-030396-8
- “The C Programming Language (Second Edition)” Brian W. Kernigham, Dennis M. Ritchie. Prentice Hall Software Series. 1988. ISBN 0-13-110370-9
- “Expert C Programming - Deep C Secrets”, Peter Van Der Linden, SunSoft Press (PTR PH) A Prentice Hall Title ISBN 0-13-177429-8
- “Implementing Lightweight Threads”, D. Stein and D. Shah, Summer ‘92 USENIX.
- Solaris 2 Late Breaking News (release notes) and Solaris 2 Answerbooks

Internet Resources

- ftp://opcom.sun.ca/pub - For many of the tools referenced
- http://www.Sun.COM/smcc/solaris-migration - Solaris Migration page
- http://christensen.cybernetics.net/solaris.html - Free Software ported to Solaris
- ftp://sunsite.unc.edu/ - public FTP server with lots of sun related stuff
- ftp://prep.ai.mit.edu/pub/gnu - GNU software
- ftp://cs.uiuc.edu/pub/xemacs - XEmacs editor
- ftp://research.att.com/dist/sam - Sam editor
- ftp://ds.internic.net/rfc - Internet RFCs
- ftp://rtfm.mit.edu - UseNet FAQs (Solaris 2 and Solaris 2 Porting)