Porting SCO OpenServer 5 Applications to Release 6

August 22, 2006

Information in this document is subject to change without notice and does not represent a commitment on the part of The SCO Group, Inc.

Document Version: SCO OpenServer Release 6.0.0

September 2005

Contents

1 Porting SCO OpenServer 5 Applications to SCO OpenServer 6

This document describes porting SCO OpenServer Release 5 applications to Release 6, and techniques that can be used to make such a port successful.

1.1 About porting SCO OpenServer 5 applications

Porting applications to SCO OpenServer 6 is not difficult. Many ports will involve no more than making a few changes to a makefile, then recompiling using the SCO OpenServer 6 Development System (Dev Sys).

Porting may not even be necessary:

- An application that uses only SCO-supported Release 5 interfaces should run unmodified on Release 6.
- An application that uses third-party interfaces not supported by SCO may still run unmodified on Release 6 if all required libraries are installed on the Release 6 system along with the application.

But you may want to recompile your application on SCO OpenServer 6 to take advantage of new features such as multithreading or large files, to gain better runtime performance, to access new resources on Release 6, or for other reasons.

The SCO OpenServer 6 Development System allows you to compile programs using either the Release 5 ABI or the Release 6 ABI, as described in the following section.

1.1.1 Compiler ABI modes: UDK and OSR5

SCO OpenServer 6 is designed to run both SCO OpenServer Release 5 (OSR5) ABI applications as well as System V Release 5 (SVR5) applications. These two kinds of applications are distinct because they:

- make different kinds of system calls to the operating system
- make use of different application programming interfaces (APIs)
- embody different application binary interfaces (ABIs).

The OSR5 ABI is the interface supported by SCO OpenServer Release 5 systems, while the SVR5 ABI is the default ABI for Release 6 (and for future OpenServer development). It is also the ABI supported by the UnixWare/OpenServer Development Kit (UDK) and so is sometimes called the UDK ABI.

Within any one operating system process, every object file must have been compiled using either the OSR5 ABI or the SVR5 ABI; this applies to the application and all the libraries it uses. These ABIs cannot be mixed within a single process.

The C compiler (**cc** command), the C++ compiler (**CC** command), and a few other development tools include a **-K** option to select the ABI (or mode), with **udk** representing the SVR5 ABI and **osr** specifying the OSR5 ABI.

The default ABI used (i.e., when no **-K** option is specified) depends upon how the compilers and related tools are invoked. For example:

/usr/ccs/bin/cc	default is UDK mode; use **-K osr** for OSR5 mode (OSR5 ABI)
/osr5/usr/ccs/bin/cc	default is OSR mode; use **-K udk** for UDK mode (SVR5 ABI)

In general, which ABI you use when you port your application to Release 6, depend on the considerations listed in the following table:

Use UDK mode if:	**Use OSR5 mode if:**
• You are writing a new application, or porting an application that has never been run on an SCO platform. • Your application needs to use threads (e.g. makes calls to pthread_create()) or large files (e.g. makes calls to fopen64()) or other advanced features of the Release 6 kernel. • You would like your application binary to run on UnixWare as well as SCO OpenServer Release 6. • You are writing Java native code (JNI) • You are writing a device driver for SCO OpenServer Release 6.	• You want to minimize code changes to your application and pick up improvements made to the OSR5 APIs, or to generate better-performing code. • You are developing an application that will also run on Release 5. • Your application needs to link with Release 5 objects or libraries (*.o*, *.a*, *.so* files) that you or others created, that you do not want to (or cannot) recompile.

For more information about using these compiler modes, see the Dev Sys overview at: **http://osr600doc.sco.com/en/SDK_oview/DEVSYS.oview.html**.

1.1.2 Mixed mode issues

If you have an existing SCO OpenServer 5 application that you want to modernize with threads or large files or some other SVR5-dependent feature, you must compile it in UDK mode. Because UDK mode features some different APIs from SCO OpenServer 5, it is possible that you will have some porting work and source code modifications to do. Occasionally the mode requirements may present a contradiction: for instance, you have an existing SCO OpenServer 5 application program that depends upon and is linked against a third-party library archive. Now you want to modify another part of this program to handle large files. That means you need to use UDK mode, but the presence of the third-party library means you need to use OSR mode. Assuming you cannot get the library vendor to re-build it in UDK mode, what do you do?

In this situation you must split your application program into two processes. One process makes the calls to the third-party library and is compiled in OSR5 mode. The other process makes the calls to do the large file I/O and is compiled in UDK mode. Then you must use some sort of inter-process mechanism (such as sockets, pipes, or IPC) to communicate between the two processes.

1.2 Tool sets supported

The SCO OpenServer 6 Development System (Dev Sys) is found on SCO OpenServer 6 Installation CD-ROM in the media kit. The Dev Sys offers:

- the best integration with SCO OpenServer 6
- generally the best performing generated code (in terms of size and speed)
- a stronger debugger
- the best ability to have the resulting binaries run not just on SCO OpenServer 6, but SCO OpenServer 5 and UnixWare 7 as well.

This document assumes that the SCO OpenServer 6 Dev Sys is being used.

If your application is in Java, then the Java 2 Standard Edition for SCO UNIX Operating Systems is the vehicle to use to build (if necessary) and run the application. This is installed by default on every SCO OpenServer 6 system.

1.3 Compiler option guidelines

Basic compiler options are defined by POSIX and tend to be the same across platforms (such as **-c** and **–P**). Others are trickier and can lead to porting problems. Here are some of the more common problems:

- **Link with the right command**. Some existing makefiles from SCO OpenServer 5 will link C or C++ programs with ld or link C++ programs with **cc**, neither of which work with SCO OpenServer 6. Always use **cc** to link C programs and CC to link C++ programs (if the program is a mixture, use CC). The same is true if you are linking dynamic libraries (*.so* files). In this case, also use the **-G -KPIC** options to indicate you want a dynamic library with relocatable code. (Using the **-s** option that some systems use for this will generate a stripped library that has no debug information.) For both C and C++, much more is done during the link step than simple linking. For C, which now uses DWARF II for debug information, debug abbreviation tables are brought in. For C++, template instantiation prelinking is performed, static constructor and destructor sequences are anchored, and so on. As a rule, using **ld** directly is inappropriate except when using **ld -r** to create a conglomeration of object files.
- **Do not add -lc or -lC to the link command line**. This is sometimes present in makefiles, but the order in which system libraries are linked is critical. The **cc** and **CC** commands do this automatically – do not call these libraries explicitly (the compiler will generate a warning).

- **Use -Kthread or -Kpthread instead of -lthread**. Multithreaded applications frequently use makefiles that pass **-D_REENTRANT**, **-D__THREADSAFE**, or some other such symbol to compilers, and then link with **-lthread**. On SCO OpenServer 6, both C and C++ use a **-Kthread** (for SVR4 threads) or **-Kpthread** (for POSIX threads) option at preprocessing, compile, and link time to indicate that multithreading is active.

  Using **–lthread** will cause problems (and the compiler will generate a warning), because of the system library ordering problem discussed previously.

- **Use dynamic links, not static**. Applications work best on SCO OpenServer 6 when they are dynamically linked against system libraries instead of linked statically. This ensures the appropriate version of a library is used. (This is why there are no static versions of system libraries like **libsocket** and **libnsl**.) If you must link statically, use **-dn** and not **-Bstatic**. The latter might have the desired effect with some other compilers, but its meaning in SCO OpenServer 6 is different (it toggles the way the linker looks for libraries) and using it improperly usually causes a runtime failure.

- **Be aware of assumptions about symbol exporting**. On some UNIX platforms (notably Solaris) the linker exports all global symbols defined in an executable. This means that shared libraries will see those symbols, even if the libraries are dlopen'd during runtime. In SCO OpenServer 6, the linker only exports those symbols defined in an executable that are referenced in the shared libraries seen during the link. That means that if a library is dlopen'd during runtime, it will not see any other symbols that may have been defined in the executable. If these symbols are needed, they should be explicitly exported by passing in the **-Wl,-Bexport=*name*** option to the linker. Simply passing **-Wl,-Bexport** will cause the linker to match the Solaris default behavior.

- **Libraries for graphical apps**. For SCO OpenServer 6, there is a list of libraries that must be named to link a Motif application:

  ```
  -lXm -lXt -lXext -lX11 -lSM -lICE -lsocket -lnsl
  ```

  Most makefiles from other compilers do not have all these libraries named. If they are not specified, unresolved symbols will result.

- **Libraries for networking apps**. For SCO OpenServer 6, the order of networking libraries is significant. It should be done like this:

  ```
  [ -lresolv ] -lsocket -lnsl
  ```

  This means libresolv need not be present, but should go first if it is used. The other two are mandatory and should be used in the order shown.

1.4 API issues

In general, SCO OpenServer 6 is compliant with POSIX, XPG4, UNIX 95, and UNIX 98. If the application being ported is also compliant to these standards, a simple recompile and relink should be all that is necessary. In reality, things are rarely that easy. Kernel and API compatibility issues may affect the ability of your application to run as expected on Release 6. These issues are described in the section "Kernel and API Compatibility Notes" on page 23.

1.5 C language dialect issues

- Generally, the Dev Sys C compiler accepts any ISO-standard C code. In default mode, this also includes accepting K&R C dialect (but using ISO semantics where there is a difference). You can give those cases K&R semantics using the **cc -Xt** transition compilation option. This is usually done to preserve "unsigned-preserving" integral promotion rules (versus the standard "value-preserving" integral promotion rules). Do not use **cc -Xt** unless you must. It is better to bring code up to date since using **cc -Xt** can result in a significant performance loss (due to the lack of the volatile attribute).

- Another porting problem occurs for code that assumes a particular signedness of "plain" `char`. By default, the Dev Sys compiler takes them as `signed`; the default can be changed to `unsigned` using the **-Kuchar** option.

- The Release 5 Dev Sys C compiler's **-Xk** option for K&R mode is not supported by the Release 6 Dev Sys C compiler; use **-Xt** instead. In addition, the OSR5 C compiler's **-Xm** option for 'Microsoft mode' is obsolete and no longer supported.

- Note that the Dev Sys C compiler fully supports 64-bit integers via the `long long` type.

- The most frequent source of C dialect problems comes from gcc-built applications. The GNU C compiler has a number of extensions in it that are not supported on other platforms. The most well-known of these is support for variable-sized arrays:

  ```
  int length(void);
  int foo(void)
  {
       int x=length();
       char f[x];
       ...
  }
  ```

  This feature is present in a somewhat different form in the ANSI/ISO 1999 C standard, but is not supported in the Dev Sys C compiler. This means the code must be rewritten using heap allocation.

- Another change brought about by the C 99 standard is that inline and restrict are now C keywords. If this conflicts with existing code and it is not practical to modify the code, the Dev Sys **cc -Xb** option can be used to suppress the recognition of these two new keywords.

1.6 C++ language dialect issues

The ANSI/ISO 1998 C++ standard invalidated a lot of old, existing C++ code. The following are a few of the better-known examples:

- new keywords

- for loop variable scope change

- template specialization syntax, guiding declarations
- template friends syntax change
- new failure throws exception
- implicit int gone

A more detailed discussion of some of these incompatibilities can be found in newer C++ textbooks, such as Bjarne Stroustrup's *The C++ Programming Language* (Third Edition), or in the Internet *McCluskey C++ Newsletter.*

So how do you get existing C++ code bases to compile using the Dev Sys compiler? For many years the original AT&T **cfront** was the most heavily used C++ compiler. For existing **cfront**-built code, there is a Dev Sys **CC -Xo** compatibility option that provides **cfront** source compatibility. But this also enables **cfront** bug tolerance and many anachronisms. There are some instances of **cfront** that **CC -Xo** will not detect or accept. It is intended as a transitional tool for converting code to the modern dialect of C++ acceptable by the Dev Sys C++ compiler. It should not be used on a permanent basis.

For example, consider this code:

```
template <class T>
class A {
     int m, n;
     int explicit;
public:
     void f();
     T g();
};
template <class T>
void A<T>::f() {
     for (int i = 0; i < 10; i++)
          n += i;
     for (i = 0; i < 10; i++)
          m += i;
}
char A<char>::g() { return 'a'; }
int main() {
     A<int> a;
     a.f();
}
```

This compiles under a cfront-era C++ compiler without problems, but generates many errors with the Dev Sys C++ compiler. The same code source compiles without error under the Dev Sys C++ compiler using the **-Xo** option.

Most ANSI/ISO source language incompatibilities are not complicated and can be straightforwardly resolved, so in most cases it is best to bring the code up to date. In this case, it's simply a matter of avoiding the new keyword, using the new template specialization syntax, and adjusting to the new for loop variable scope rule.

Here is the new version with three simple edits that compiles properly on the OSR6 Dev Sys C++

compiler without using any compatibility option:

```
template <class T>
class A {
     int m, n;
     int is_explicit; // changed
public:
     void f();
     T g();
};
template <class T>
void A<T>::f() {
     for (int i = 0; i < 10; i++)
          n += i;
     for (int i = 0; i < 10; i++) // changed
          m += i;
}
template<> char A<char>::g() { return 'a'; } // changed
int main() {
     A<int> a;
     a.f();
}
```

1.7 Changes to the runtime environment

The subsections that follow discuss changes to the runtime environment in SCO OpenServer 6 that must be taken into account when porting any Release 5 application. Your application may also encounter runtime issues related to other changes in the runtime environment between Release 5 and Release 6. See the *Upgrade Guide* at **http://www.sco.com/support/docs/openserver/600/migration** for more information on changes to the runtime environment in Release 6.

> **Note**: While the SVR5 ABI is the default runtime binary interface, the SVR5 kernel recognizes OSR5 ABI applications and runs them in the environment that OSR5 ABI applications expect. There are no special steps or commands needed to run an application on Release 6 that was compiled in OSR5 mode (or compiled using the Release 5 Development System).

1.7.1 Console Display Problems

Some SCO OpenServer 5, SCO Unix, and SCO Xenix applications may not display correctly on the OpenServer 6 console, even though they ran without problems on the SCO OpenServer 5 **scoansi** console. These problems include:

- Line drawing characters are replaced by accented characters.
- Function keys do not return the expected values.

The most likely cause of these problems are poorly written applications that are using hardwired escape sequences. There also may be problems caused by a clash between the codeset that the application is using and that used by the OpenServer 6.0.0 console.

There are a number of possible solutions to these problems. For more information, please go to the

SCO Support Knowledge Center (**http://wdb1.sco.com/kb/search/**) and search for TA# **126154**.

1.7.2 Device information

Some OSR5 ABI applications parse various system data files on Release 5 (such as */dev/string/cfg*) to obtain device information, such as SCSI addresses. These methods in general should not be used on OpenServer 6, since many (such as */dev/string/cfg*) no longer exist, and the format of these files changes as new technologies are introduced. Similarly, output formats for some device-related commands have and will continue to change (such as **sdiconfig -l**).

The **devattr** command is the reliable way to obtain device information on Release 6 and future releases:

```
# devattr -v cdrom1
alias='cdrom1'
bdevice='/dev/cdrom/c1b0t0l0'
bdevlist='/dev/cd0,/dev/cdrom/cdrom0'
bklib='SCSI'
bufsize='2048'
cdevice='/dev/rcdrom/c1b0t0l0'
cdevlist='/dev/rcd0,/dev/rcdrom/cdrom0'
desc='SCSI CD-Rom Drive 1'
display='true'
inquiry='TOSHIBA DVD-ROM SD-M17121004'
mountpt='/installr'
pdimkdtab='true'
removable='true'
scsi='true'
type='cdrom'
volume='CD'
```

Use the **getdev** command to list the device aliases to use with **devattr**:

```
# getdev | pg
cdrom1
disk1
diskette1
hba1
hba2
...
```

If the **cdrtools** package is installed, you can also use the **cdrecord -scanbus** command to scan the SCSI bus:

```
# cdrecord -scanbus
Cdrecord-ProDVD-Clone 2.01 (i586-sco-sysv5uw7.1.4) Copyright (C) 1995-2004 Jörg
Schilling
Using libscg version 'schily-0.8'.
scsibus1:
        1,0,0   100) 'TOSHIBA ' 'DVD-ROM SD-M1712' '1004' Removable CD-ROM
        1,1,0   101) *
        1,2,0   102) HOST ADAPTOR
        1,3,0   103) *
        1,4,0   104) *
        1,5,0   105) *
        1,6,0   106) *
```

```
        1,7,0   107) *
```

1.7.3 Runtime commands

If a Release 5 application requires Release 5 runtime commands (that is, it fails using the default command reached by the default PATH variable on OpenServer 6), Release 5 commands are supplied in the directory */osr5/bin*. Change the PATH variable for the application so that it accesses binaries under */osr5/bin* before other command directories. Manual pages for these commands can be found on the OpenServer 507 doc web site: **http://osr507doc.sco.com**.

1.7.4 Runtime access of system/data files

Although links have been used to minimize problems associated with changes of location of system files accessed by applications at runtime, some applications that depend on the internal format of files may experience problems interpreting file contents. One example is the default */bin/cpio* command, which uses an "Extended cpio File Format", instead of the "ASCII cpio format" used by the **cpio** command on OpenServer 5. If your application requires the OpenServer 5 cpio file format, then you'll need to change your application to access the */osr5/bin/cpio* version of **cpio** as described in the last section.

1.7.5 NETbios/NETbeui

NETbios/NETbeui are no longer supported; runtime calls to NETbios commands and references to NETbios resources will fail on SCO OpenServer Release 6.

1.7.6 XENIX 286 emulators

Calls to the legacy **i286emul**/**x286emul** runtime emulators and references to associated resources will fail on OpenServer 6.

1.8 Binary debugging

SCO OpenServer Release 6 includes a number of tools for debugging. The sections that follow discuss some of these features in more detail.

debug	Powerful and effective tool for source- and assembly-language debugging. Replaces **dbx**, **dbXtra**, and other OpenServer 5 debuggers. Offers both graphical and command-line user interfaces. Full support for both C and C++. Full support for multi-process and multi-thread debugging. Available in both UDK and OSR5 modes (multi-thread does not apply to OSR mode).
fprof	Flow profiling to analyze performance and locality of reference of text. Only available in UDK mode, but similar functions are available with **fur**.

fur	Object-level optimizer that uses dynamic feedback from flow profiling to transparently rearrange application object code for better performance. Available in both UDK and OSR5 modes. While **fprof** is not available in OSR5 mode, fur can perform instrumentation and flow analysis at the code block level. Based on the flow through each function, **fur** can reorder the object code in a function to: • improve reference locality • reduce instruction execution stalls • reduce branching • improve cache hits as well as reorder functions within an object.
memtool	Memory error detection tool that finds common C or C++ memory errors and leaks and gives source-level information about them. Only available in UDK mode.

1.8.1 Tracing system calls with truss

SCO OpenServer 6 includes a useful command named **truss** that can trace the system calls made by an application. Using truss does not require recompilation, relinking, access to the source code, or even a symbol table. The **truss** command replaces the Release 5 **trace** and **scotruss** commands.

Consider this code:

```
$ cat fstab.c
#include <stdio.h>
int main() {
     FILE* fp = fopen(“/etc/fstab”, “r”);
     if (!fp)
          fprintf(stderr, “*** cannot open file\n”);
     fclose(fp);
}
$ cc fstab.c
$ ./a.out
*** cannot open file
```

SCO OpenServer 6

The program fails because SCO OpenServer 6 uses */etc/vfstab* not */etc/fstab*, but the error message does not indicate the file it failed to open. Running the program through **truss** displays each system call, along with its arguments and the return code:

```
$ truss ./a.out
execve(“./a.out”, 0x08047570, 0x08047578) argc = 1
open(“/etc/fstab”, O_RDONLY, 0666) Err#2
ENOENT
*** cannot open file
write(2, “ * * * c a n n o t  o”.., 21) = 21
```

```
_exit(-1)
```

In a large program (especially one you are not that familiar with), **truss** can help you determine the point of failure. Using the **–f** option, truss can follow child processes as well as the parent process. You can also use truss to grab and release existing processes on the system.

Along with knowing symbolic names for many manifest constants (the O_RDONLY above, for example), truss can display additional information to further focus the trace. An explicit list of system calls to trace (or not to trace) can be given to truss with the **-t** option.

The **-a** and **-e** options respectively cause truss to display all the passed arguments and the entire environment at an exec.

Another advantage is that truss knows structures that cross the system call boundary. Using the **-v** option, you can specify system calls for which you want to see complete structures displayed.

Finally, all of the data read and/or written on a file descriptor (instead of the default first few bytes) can be displayed using the **-r** and/or **-w** options.

```
$ truss -w2 ./a.out
execve("./a.out", 0x08047570, 0x08047578) argc = 1
open("/etc/fstab", O_RDONLY, 0666) Err#2
ENOENT
*** cannot open file
write(2, 0x08047C14, 21) = 21
     * * * c a n n o t  o p e n  f i l e\n
_exit(-1)
```

Note that you must be *root* to run **truss** on privileged programs.

1.8.2 Using the dynamic library tracing feature of the runtime linker

A feature similar to **truss** is provided for dynamic executables by the runtime linker in SCO OpenServer 6.

When a dynamic executable calls a function that (potentially) is defined elsewhere -- outside of the calling shared library or executable -- a call is actually made to a bit of code in the "procedure linkage table" (PLT). This code then does an indirect branch to the target function. But, at process startup (by default), all the PLT entries end up branching to the runtime linker, which searches the executable and the current set of shared libraries for the actual definition of the target function. Once found, the runtime linker modifies the destination of the indirect branch for that PLT entry so that it jumps to the actual function.

Dynamic library tracing takes advantage of this lookup process and interposes a reporting function between the caller and the callee. It can also insert a bit of code to return to that comes between the callee and the caller that similarly reports the return value of the function.

As with truss, dynamic library tracing is disabled for processes that gain privilege for obvious secu-

rity reasons.

This tracing differs from truss in a number of ways:

- Because each function can call further functions, the call and return are displayed separately.
- The runtime linker only knows a name to search for and (when found) the address. It does not inherently understand anything else about a function, while truss has knowledge of each system call. However, if the defining library provides a special table of encoded bits with an entry for this function, the reporting code will be able to do something other than just print a few hexadecimal words that might be actual arguments. Only the C and threads libraries provide this table. (Note that if an entry is found, the reporting code can cause the process to die if it attempts to dereference a bad string pointer.)
- The runtime linker can only report functions that go through its lookup processing (internal function calls are not reported).
- The reporting code also displays the caller of each function.

Dynamic library tracing is enabled by running a process with certain environment variables set:

LD_TRACE	Without LD_TRACE in the environment, no reporting will occur. If it exists but has an empty string value, lines of this form are displayed: `sym(arg1,arg2,arg3) from hex_addr` For unknown functions, three hexadecimal words are printed as arguments. Otherwise, the value of LD_TRACE is expected to be one or more comma-separated keywords from the following list: • **sym** -- display name+offset instead of hex_addr. The +offset part is only shown when nonzero. • **lib** -- add @lib after a hex_addr or name+offset to give the containing library (or executable). • **ret** -- also include tracing function returns. • **tim** -- add a at sec.usec. • **hitim** -- add either a at ticks or, when LD_TRACE_SCALE is set, at sec.ticks. Note that the high-resolution timer requires the Pentium rdtsc instruction. Moreover, using this instruction is a privileged operation by default. To enable this feature in SCO OpenServer 6, you must set USER_RDTSC to nonzero using idtune and rebuild and reboot. Finally, because we do not synchronize the processor timers on multiple CPU boxes, the timings are likely to be inaccurate for such systems.

LD_TRACE_FILENO	Provides the file descriptor to which to write the reports. By default, it is 2 (standard error).
LD_TRACE_ARGNO	Provides the number of hexadecimal words to display for unknown functions. It is 3 by default.
LD_TRACE_SCALE	Provides the clock speed in MHZ to be used with hitim.
LD_TRACE_STACK	Provides a simple stack trace feature. Its value is expected to be a comma-separated list of function names for which to display a stack trace at each call. All other reporting is disabled when LD_TRACE_STACK is set. The stack trace algorithm uses a fairly straightforward scheme which, unfortunately, fails for functions that return structures.
LD_TRACE_FRAMES	Provides a maximum stack depth to trace when LD_TRACE_STACK is set. The default value is zero which is taken to mean the entire stack.
LD_TRACE_ROUTINE LD_TRACE_LIBRARY	Provide comma-separated lists of specific routine names and library names, respectively, to report. Both of these interpret a leading "!" as inverting the list (all but ...). Note that LD_TRACE_STACK wins over LD_TRACE_ROUTINE. These two variables can be used to greatly trim down the large amount of noise generated by default with LD_TRACE.
LD_TRACE_MAXSTR	Sets the limit on string printing to a value in the range [0,120], with the default being 20. (A value of 0 disables the dereferencing, which may be necessary when a bad pointer has been passed to a routine that RTLD expects to display as a string.)

Tracing the same *fstab.c* example from above:

```
$ LD_TRACE=sym
./a.out
atexit(0x80483b4) from 0x8048459
atexit(0xbff9b09c) from 0x8048465
atexit(0x8048514) from 0x804846f
__fpstart() from 0x804847b
fopen("/etc/fstab","r") from 0x80484b1
_findiop() from _realfopen+7
_open("/etc/fstab",0x0,0x1b6) from endopen+141
__thr_errno() from _cerror+29
fprintf(0x80495ec,"*** cannot open
file"...,0x0,0x8047cb8,0x8048485) from 0x80484cc
_findbuf(0x80495ec) from fprintf+116
_xflsbuf(0x8047c44) from _idoprnt+358
_write(2,0x8047c04,21) from _xflsbuf+89
*** cannot open file
fclose(0x0) from 0x80484d7
exit(-1) from 0x804848e
_exithandle() from exit+18
_setjmp(0xbfffe628) from _exithandle+100
_cleanup() from _exithandle+151
fflush(0x0) from _cleanup+9
```

Note that even with symbols enabled, the reporting of some addresses (like the caller of fopen) is still just a hexadecimal value. This is because the only names available to the reporting code are those exported from the executable and shared libraries and by default only the necessary few global names are exported by the linker in the executable. If you use pass the **-Wl,-Bexport** option to the linker (see above regarding exported names) all global functions are exported.

Adding return value tracing:

```
$ LD_TRACE=sym,ret ./a.out
atexit(0x80483b4) from 0x8048459
     => atexit returned 0
atexit(0xbff9b09c) from 0x8048465
     => atexit returned 0
atexit(0x8048514) from 0x804846f
     => atexit returned 0
__fpstart() from 0x804847b
     => __fpstart returned
fopen("/etc/fstab","r") from 0x80484b1
_findiop() from _realfopen+7
     => _findiop returned 0x80496d0
_open("/etc/fstab",0x0,0x1b6) from endopen+141
__thr_errno() from _cerror+29
     => __thr_errno returned 0xbfffedd0
     => _open returned -1
     => fopen returned 0x0
fprintf(0x80495ec,"*** cannot open
file"...,0x0,0x8047cb8,0x8048485) from 0x80484cc
_findbuf(0x80495ec) from fprintf+116
     => _findbuf returned 0x80495ec
_xflsbuf(0x8047c44) from _idoprnt+358
_write(2,0x8047c04,21) from _xflsbuf+89
*** cannot open file
     => _write returned 21
     => _xflsbuf returned 0
     => fprintf returned 21
fclose(0x0) from 0x80484d7
     => fclose returned -1
exit(-1) from 0x804848e
_exithandle() from exit+18
_setjmp(0xbfffe628) from _exithandle+100
_cleanup() from _exithandle+151
fflush(0x0) from _cleanup+9
     => fflush returned 0
     => _cleanup returned
     => _exithandle returned
```

Note in this example that the return from setjmp is not displayed. Certain special functions like setjmp are sensitive to their return address. Finally, a stack trace for write results in the following:

```
$ LD_TRACE=sym LD_TRACE_STACK=_write
./a.out
[0] _write(2,0x8047bec,21)
[1] _xflsbuf+89(0x8047c2c) [0xbffa7b05]
[2] _idoprnt+358(0x8047c2c,0x80484f8,0x8047c88) [0xbffa1eb6]
[3] fprintf+201(0x80495ec,"*** cannot open
file"...,0x0,0x8047ca0,0x8048485) [0xbffa8195]
[4] 0x80484cc(0x1,0x8047cac,0x8047cb4)
Page 41 SCO OpenServer 6
```

```
[5] 0x80484cc(0x8047cac,0x8047cb4,0x0)
*** cannot open file
```

1.8.3 Memory debugging with memtool

C (and to some degree C++) programs are infamous for memory allocation/corruption problems that are difficult to debug. These can sometimes show up during ports of "working" code because a malloc/free mistake might be missed on one platform but not on another. Such problems can go undetected for some time. Commercial tools such as Purify can help track down such problems.

The Dev Sys has a similar tool created by SCO, called **memtool**. Consider this simple error-ridden program:

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4
5 int main() {
6    char *p = malloc(7), *q = "yes, too long\n";
7    strcpy(p, q);
8    free(p);
9    q = 0;
10   *p = 'Y';
11   realloc(p, 3);
12   fputs(p, stderr);
13   return *q;
14 }
```

Most of the time, this program will run without problems, but will occassionally fail with memory allocation errors. **memtool** can be used to pinpoint the code in your application that may cause memory-allocation-related problems. If you take the same program as above and recompile it with **-g** for maximum symbolic information, you can then run the executable using **memtool**. This produces the following diagnostic output:

```
$ cc -g error.c
$ /usr/ccs/bin/memtool ./a.out
================================================================================
Some abuses or potential misuses of the dynamic memory allocation subsystem
have been detected.  The following multiline diagnostics are descriptions
of what exactly has been seen.  They may include use of terms and concepts
with which you may be unfamiliar.  Use the "-v" option to get the complete
version of this introduction.
================================================================================
A block's spare bytes have been modified.  This usually occurs due to
writing beyond the block's regular bytes, either because an insufficient
number of bytes were requested when the block was allocated or simply
due to a programming logic error.

History for block at address 0x8049b70:
*Stack trace when detected:
  [0] free(ptr=0x8049b70)
  [1] main(0x1,0x8047e24,0x8047e2c) [error.c@8] in ./a.out
  [2] _start() [0x80485f4] in ./a.out
```

```
*Stack trace when block at 0x8049b70 was allocated:
  [0] malloc(sz=7)
  [1] main() [error.c@6] in ./a.out
  [2] _start(presumed:0x1,0x8047e24,0x8047e2c) [0x80485f9] in ./a.out

Annotated snapshot surrounding the live allocation at address 0x8049b70
when the 5 bytes [0x8049b77,0x8049b7b] were found to have been modified.
This allocation holds 7 byte(s) followed by 5 extra (or spare) byte(s),
and, in this case, spare bytes were found to have been modified.

0x8049b60: 0x00000000 0x00000000 0x00000000 0x00000011  ................
         :                                   ********              ****
0x8049b70: 0x2c736579 0x6f6f7420 0x6e6f6c20 0x00000a67  yes, too long...
         :   --------   ^^------   ^^^^^^^^             -------^^^^^

================================================================================
A block's header has been modified.  Often this occurs due to mistakenly
writing past the end of the preceding block.  You might also try using the
"-x" option to add spare bytes to the end of each block, and see whether
your application behaves differently.

History for block at address 0x8049b80:
*Stack trace when detected:
  [0] free(ptr=0x8049b70)
  [1] main(0x1,0x8047e24,0x8047e2c) [error.c@8] in ./a.out
  [2] _start() [0x80485f4] in ./a.out

Annotated snapshot surrounding the live allocation at address 0x8049b80
when the 4 bytes [0x8049b7c,0x8049b7f] were found to have been modified.
This allocation holds 4 bytes, but, in this case, the allocation's
header was found to have been modified.

0x8049b70: 0x2c736579 0xca6f7420 0xcacacaca 0x00000a67  yes, to.....g...
         :                                   ^^^^^^^^              ^^^^
0x8049b80: 0x00000000 0x00000015 0xcacacaca 0xcacacaca  ................
         :   --------                                  ----

================================================================================
A recently free()d block was passed as the first argument to realloc().
Only null pointers or live block addresses are permitted to be passed to
realloc(), although, in this implementation, were dynamic memory checking
not enabled, this block's contents would have been preserved between its
being freed and this call to realloc(), but this is a nonportable feature
of this implementation which should not be relied on.

History for block at address 0x8049b70:
*Stack trace when detected:
  [0] realloc(ptr=0x8049b70,sz=3)
  [1] main(0x1,0x8047e24,0x8047e2c) [error.c@11] in ./a.out
  [2] _start() [0x80485f4] in ./a.out

*Stack trace when block at 0x8049b70 was released:
  [0] free(ptr=0x8049b70)
  [1] main() [error.c@8] in ./a.out
  [2] _start(presumed:0x1,0x8047e24,0x8047e2c) [0x80485f9] in ./a.out

*Stack trace when block at 0x8049b70 was allocated:
  [0] malloc(sz=7)
  [1] main() [error.c@6] in ./a.out
```

```
  [2] _start(presumed:0x1,0x8047e24,0x8047e2c) [0x80485f9] in ./a.out

================================================================================
A free()d block has been modified.  This usually means that the block was
passed to free() or realloc(), but the block continued to be used by your
application, possibly far removed from the deallocating code.

History for block at address 0x8049b70:
*Stack trace when detected:
  [0] realloc(ptr=0x8049b70,sz=3)
  [1] main(0x1,0x8047e24,0x8047e2c) [error.c@11] in ./a.out
  [2] _start() [0x80485f4] in ./a.out

*Stack trace when block at 0x8049b70 was released:
  [0] free(ptr=0x8049b70)
  [1] main() [error.c@8] in ./a.out
  [2] _start(presumed:0x1,0x8047e24,0x8047e2c) [0x80485f9] in ./a.out

*Stack trace when block at 0x8049b70 was allocated:
  [0] malloc(sz=7)
  [1] main() [error.c@6] in ./a.out
  [2] _start(presumed:0x1,0x8047e24,0x8047e2c) [0x80485f9] in ./a.out

Annotated snapshot surrounding the free allocation at address 0x8049b70
when the byte at 0x8049b70 was found to have been modified.  This
allocation holds 12 bytes, one of which was found to have been modified.

0x8049b60: 0x00000000 0x00000000 0x00000000 0x00000011  ................
         :                                   ********             ****
0x8049b70: 0xcacaca59 0xcacacaca 0xcacacaca 0x00000179  Y...........y...
         :   ------^^   --------   --------             ^-----------

================================================================================
          LIVE ALLOCATIONS AT PROCESS EXIT
================================================================================
Memory allocations that have not been released by the time your process is
finished are in no way an error, but they are potentially ``leaks'' --
allocations that inadvertently have not been released once they were no
longer needed or in use.  If your application has more than a few live
allocations displayed below with the same size and/or were allocated at the
same location, you may well have a leak; or if your application, when run
with more data or for longer periods, displays more live allocations here,
you also may have a leak.  A leak also need not be repaired:  A short-lived
process can easily survive having many leaks, as long as they are not too
large and there are not so many that available memory resources could become
exhausted (or even strained).  Moreover, it may well be that ``the leaks''
are allocations that were all in use up to just before the process exits,
and the effort and the expense to release them all is not warranted.


Stack trace for 4 byte block at 0x8049cf8:
  [0] realloc(ptr=0x8049b70,sz=3)
  [1] main() [error.c@11] in ./a.out
  [2] _start(presumed:0x1,0x8047e24,0x8047e2c) [0x80485f9] in ./a.out

*Stack trace for block previously at 0x8049b70 before realloc() to 0x8049cf8:
  [0] free(ptr=0x8049b70)
```

```
[1] main() [error.c@8] in ./a.out
[2] _start(presumed:0x1,0x8047e24,0x8047e2c) [0x80485f9] in ./a.out
```

1.8.4 Source debugging with debug

The Dev Sys debugger, **debug**, is a powerful tool for source and assembly-level debugging. It supports both C and C++ language debugging. (For Java debugging, use the command-line jdb debugger.) The debugger has two basic user interfaces: command line and graphical.

Debugging the port of an existing application can be different from program development. Some debugger features that may be useful in this regard are:

- The **map** command is excellent at showing the layout of memory, and where any arbitrary address might be.
- Multithreaded debugging and multiprocess debugging support is strong in the debugger. Many commands take a **-p process-nbr.thread-nbr** argument indicating a specific process or thread, with **-p** all applying to all processes or threads.
- You can follow program forks into both the parent and child processes; other debuggers only allow one or the other.
- The debugger allows you to grab existing processes (for example, that are hung in loops) or debug core dumps.

The **debug** tool is not related to the **dbx** and **gdb** debuggers many Release 5 users are familiar with, and uses a different command syntax. However, there is “A Guide to debug for dbx Users” on page 63 that will help such users gain familiarity with **debug**.

For more information on the Dev Sys debugger, see the Dev Sys **debug** documentation, on the web at **http://osr600doc.sco.com/en/SDK_cdebug/CONTENTS.html**.

1.9 Development System documentation

See the SCO OpenServer 6 Development System Documentation on the web at **http://osr600doc.sco.com/en/Navpages/SDKhome.html**. If you installed the Development System on your OpenServer 6 system, the documentation is also available locally at **http://*hostname*:8457/en/Navpages/SDKhome.html**, where ***hostname*** is the name of your system or **localhost**.

2 Kernel and API Compatibility Notes

As discussed earlier in this guide, SCO OpenServer Release 6 supports two ABIs: OSR5 and SVR5. In addition, the common execution environment provided by the SCO OpenServer Release 6 kernel and runtime libraries permits you to run most existing SCO OpenServer and UnixWare 7 applications without modification.

As an aid to porting applications, the differences between the two ABIs are discussed in detail in "Kernel compatibility" on page 23 and "API Compatibility" on page 35.

The more general subject of using the SVR5 libraries, compilers, header files, and other tools on any platform is documented in the Software development documentation at **http://osr600doc.sco.com/en/Navpages/SDKhome.html**.

2.1 Kernel compatibility

This section details the differences between system calls, libraries, ioctls, system files, and commands supported on SCO OpenServer Release 5 and Release 6.

2.1.1 Executable formats

This section deals with the kernel's ability to recognize and start the load process for binaries of various types. This includes setting compatibility switches so the kernel can make decisions based on that knowledge when processing various system calls and libray routines.

The Release 5 Development System marks Executable and Linking Format (ELF) binaries in a distinctive way; the Release 6 **cc** command with the **-K osr** option marks binaries similarly. The Release 6 kernel uses this information to switch on Release 5 system call semantics for OSR5 ABI binaries. (Note that COFF executables are also assumed to be OSR5 ABI binaries.)

When the Release 6 kernel loads an ELF executable, the ELF processing code looks at the ELF note section of the executable and, if it is 28 bytes long, it assumes the executable is a Release 5 ELF executable.

Release 6 also provides a command that specifically mark the ELF flags section with a special string; the Release 6 kernel checks for this string in the flag section and, if present, assumes the executable is a Release 5 ELF executable. Using **elfmark**(CP), any ELF binary can be marked as an OSR5 ABI binary.

2.1.2 Error numbers

There are two incompatibilities in the error numbers returned by Release 5 and Release 6 in `errno`:

- Some errors have different error numbers on each system (see "Errors with different values" on page 24)
- The same calls on each system can generate errors that are not generated on the other system (see "System calls with different error returns" on page 26)

2.1.2.1 Errors with different values

There are three classs of error numbers with different values on Release 6 and Release 5:

- Errors that are semantically the same on both systems but have different error numbers on each system; these errors are directly translated on Release 6. See ‘‘Errors that have different errnos on each system’’.
- Errors on Release 6 that are not on Release 5; these errors are translated by the code for each system call on Release 6, so that an error appropriate for OSR5 ABI applications is returned. Applications running on Release 5 will not see these errors; the behavior of portable programs should not be made to depend on the target system returning these errors. See ‘‘Errors on Release 6 but not Release 5’’.
- Errors on Release 5 that are not on Release 6; the resolution of the incompatibilities varies depending on the system call, and is presented in ‘‘Errors on Release 5 but not SCO OpenServer 6 Release 6’’.

Errors that have different error numbers on each system.

Error	Release 6 `errno`	Release 5 `errno`
ELOOP	90	150
ERESTART	91	152
ESTRPIPE	92	153
ENOTEMPTY	93	145
ENOSTOCK	95	93
EDESTADDRREQ	96	94
EMSGSIZE	97	95
EPROTOTYPE	98	96
ENOPROTOOPT	99	118
EPROTONOSUPPORT	120	97
ESOCKTNOSUPPORT	121	98
EOPNOTSUPP	122	99
EPFNOSUPPORT	123	100
EAFNOSUPPORT	124	101
EADDRINUSE	125	102
EADDRNOTAVAIL	126	103

Errors that have different error numbers on each system.

Error	Release 6 `errno`	Release 5 `errno`
ENETDOWN	127	104
ENETUNREACH	128	105
ENETRESET	129	106
ECONNABORTED	130	107
ECONNRESET	131	108
ENOBUFS	132	109
EISCONN	133	110
ENOTCONN	134	111
ESHUTDOWN	143	112
ETOOMANYREFS	144	113
ETIMEDOUT	145	114
ECONNREFUSED	146	115
EHOSTDOWN	147	116
EHOSTUNREACH	148	117
EWOULDBLOCK	11(EAGAIN)	90
EALREADY	149	92
EINPPROGRESS	150	91

Errors on Release 6 and *not* Release 5

Error	Release 6 `errno`
ECLNRACE	59
ENOLOAD	152
ERELOC	153
ENOMATCH	154

Errors on Release 6 and *not* Release 5

Error	Release 6 `errno`
EBADVER	156
ECONFIG	157
ECANCELED	158
EUSERS	94
ENOTAUTH	160
ELKBUSY	170
Olivetti	200-223

Errors on Release 5 and *not* on Release 6

Error	Release 5 `errno`
ELBIN	75
EDOTDOT	76

2.1.2.2 System calls with different error returns

The table below shows the errors returned for various system calls that are different between the SVR5 ABI and Release 5. See the for each system call listed for the detailed resolution of the error incompatibility (if you are viewing this document online, click on the system call name in the left-most column to go to the appropriate table).

System call	Additional errors on Release 6	Additional errors on Release 5
`access`	EFAULT, ELOOP	ETXTBSY
`chdir`	EIO	
`chroot`	EACCESS, ELOOP, ENAMETOOLONG	
`creat`	ELOOP, ENAMETOOLONG	
`exec`	ENOLOAD, ENOTDIR	ETXTBSY

System call	Additional errors on Release 6	Additional errors on Release 5
`fchdir`	EIO	
`fcntl`	EFAULT, EIO, EINVAL, EOVERFLOW	
`fork`	EINTR	
`fsync`	ENOLINK	
`ftruncate`		EAGAIN
`link`	ENAMETOOLONG	
`lseek`	ENOSYS	
`mkdir`	ENAMETOOLONG, ENOSPC	
`mknod`	EINVAL, ELOOP, ENAMETOOLONG	
`mount`	EACCESS, ENOLOAD, ELOOP, ENAMETOOLONG	
`msgctl(ipcmsg)`	EOVERFLOW	
`msgsnd/msgrcv(ipcmsg)`	EINTR,EIDRM	
`nice`	EINVAL	
`ptrace`		EINVAL, EPERM
`readlink`	ENOSYS	
`rmdir`	ELOOP, ENAMETOOLONG	EINVAL
`semctl (semsys)`	EOVERFLOW, EFAULT	
`semop (semsys)`	EINTR	
`shmctl (shmsys)`		ENOSYS
`shmget (shmsys)`		ENOSYS
`symlink`	ENOSYS, EIO	EINVAL
`sysacct`	ELOOP, ENAMETOOLONG	

System call	Additional errors on Release 6	Additional errors on Release 5
umount	ELOOP, ENAMETOOLONG	
unlink	EFAULT, ELOOP	ETXTBSY
utime	EFAULT, ELOOP	

2.1.3 Signals

The signal incompatibilities between the SVR5 ABI and OSR5 ABI are managed by the kernel.

When a `sigaction` type signal is sent to a process running an OSR5 ABI binary, the kernel puts the signal data in the OSR5 ABI structure before sending it to the application.

The incompatibilities arise in the `siginfo_t` and `ucontext_t` structures, which are different. See the following sections for more detail:

• “setcontext(2)” on page 32

• “waitid(2)” on page 34

2.1.4 Core file generation

Core files generated on Release 5 systems are given file names of the form *core.pid*, where *pid* is the process ID of the process causing the core dump. On Release 6, core files are named simply *core* (though this behavior can be changed via a tunable). In general, portable applications should not make assumptions about core file naming on target systems.

2.1.5 CPU status information

On Release 5, CPU status information (active or inactive) could be obtained by opening the */dev/at1* device and making an *ioctl*() call. Release 6 has no such device, so ported applications must use the *p_online*() interface instead. Note that *p_online*() numbers processors starting at 0; on Release 5, processors are numbered starting at 1. Here is an example code fragment:

```
int state = p_online(processor_number, P_QUERY);
switch (state) {
case P_ONLINE:
     printf(“active “);
     break;
case P_OFFLINE:
     printf(“inactive “);
     break;
case P_DISABLE:
     printf(“disabled “);
     break;
default:
     fprintf(stderr, “%s: cannot determine processor_data[]\n”, Pgm);
     exit(1);
```

```
}
printf("\n");
```

2.1.6 C2 Security (libprot) library

2.1.6.1 OSR5 ABI Applications

The */osr5/usr/lib/libprot.so* library is provided for use by OSR5 ABI applications only.

OSR5 ABI applications executing on Release 6 can expect that all *libprot* calls will functions as in previous releases, with the exceptions shown in the table below. These calls all return ENOSYS on Release 6. Ported applications should be rewritten to use the Release 6 equivalents:

Release 5 **libprot** call	Release 6 equivalent
stopio(S-osr5)	
getluid(S-osr5)	**getuid**(S)
setluid(S-osr5)	**setuid**(S)
statpriv	**filepriv**(S)
chpriv	**filepriv**(S)

Note that **getluid** and **setluid** have no exact equivalent in the OSR6 kernel. The "login user ID" is a UID assigned the user's login process, is propagated to all child processes, and can never be reset once assigned.

2.1.6.2 SVR5 ABI Applications

SVR5 ABI applications use the SVR5 ABI library */usr/lib/libiaf.so* for Identification and Authentication, and these calls are not functional in the Release 6 kernel.

This will be resolved in a Release 6 Maintenance Pack when an SVR5 ABI version of *libprot* will be provided, as well as an updated *libiaf* that's compatible with *libprot* and its Identification and Authentication implementation.

2.1.7 System calls

The subsections that follow detail the differences between the system calls supported on Release 5 and the SVR5 ABI.

2.1.7.1 SUDS extension

These are a set of calls in a configurable module for Release 5 only. They are defined in a static library, *libsuds.a*, and are accessed through their own call gate that does nothing until the system is properly configured for these calls. The SUDS calls are unsupported on Release 6.

2.1.7.2 access(2)

Two constants defined in *unistd.h* for deriving the `amode` argument are not normally used on Release 5 (EX_OK and EFF_ONLY_OK). OSR5 ABI binaries will not normally use these values since they did not have kernel support, but since they are in *unistd.h*, it is possible for an OSR5 ABI application to use them. If such a binary runs on the SVR5 ABI, the EX_OK and EFF_ONLY_OK constants will be evaluated correctly as part of `amode`. This may cause the application to behave differently than it did on Release 5.

2.1.7.3 adjtime(2)

The Release 5 version of this system call is affected by a number of system tunable parameters that are not present in the SVR5 ABI: `update_rtc` (if set to 1, the real time clock is set to the new system time after `adjtime`); `clock_drift` (the rate at which adjustment is made in nanoseconds/second); and, `track_rtc` (keeps system clock accurate). OSR5 ABI applications that make assumptions based on the behavior of adjtime when the above parameters are set (on Release 5) may exhibit unexpected behavior with the SVR5 ABI.

2.1.7.4 execlp(2) / execvp(2)

The OS *libc* code retries up to 5 times on an ETXTBSY for the `execlp` and `execvp` system calls. The SVR5 ABI *libc* does not return ETXTBSY, so an application expecting this behavior will not see it.

2.1.7.5 fcntl(2)

The SVR5 ABI `fcntl` system call has many more commands than the Release 5 system call: F_DUP2, F_GETOWN, F_SETOWN, F_FREESP, F_RSETLK, F_RSETLKW, F_RGETLK. A portable application compiled using the SVR5 ABI should avoid using these commands, as they will fail with ENOSYS (or SIGSYS) on Release 5.

2.1.7.6 getrlimit/setrlimit(2)

On Release 5, the mnemonic for resource number 6 is RLIMIT_AS. For the SVR5 ABI, the mnemonic is RLIMIT_VMEM. However, the function of resource number 6 in both systems is exactly the same, so this does not cause a problem for OSR5 ABI binaries.Any source code brought to Release 6 from Release 5 would have to be changed (if it uses the Release 5 RLIMIT_AS mnemonic) before compiling with the SVR5 ABI compilation.

2.1.7.7 libattach/libdetach(S)

These Release 5 calls support static shared libraries for user applications; the calls are in the Release 5 kernel, but are not documented. These calls are not supported by the SVR5 ABI. A Release 5 application using this call on Release 6 will return ENOSYS for the call.

2.1.7.8 memcntl(2)

Although **memcntl** is not documented in Release 5, there is code in the kernel to support it. The call is documented in the SVR5 ABI, and is the same except for two extra subcommands available on Release 5: MC_MAPCPU and MC_MAPUBLK. Any Release 5 binary that uses these subcommands

will fail with ENOSYS when run on Release 6.

2.1.7.9 mmap(2)

The Release 5 *mmap.h* contains 4 undocumented subcommands not supported by the SVR5 ABI: MAP_PHMEM, MAP_KVMEM, MAP_ANCESTRAL, and MAP_NOEOF. An OSR5 ABI application will get an ENOSYS error return when it attempts to use these subcommands on Release 6.

2.1.7.10 mount(2)

File-system specific options can be a problem if a given `fstype` name is the same on both systems (in `sysfs`) but the options in `data_ptr` are not.

There is a difference in the `mflags=0x8` defined in *mount.h*. On Release 5 it is 'MS_CACHE', described in mount.h as "RFS client caching". On Release 6 it is 'MS_HADBAD', described in mount.h as "file system incurred a bad block so set sstate to FSBADBLK on mount".

The MS_NOEXEC mflag is defined in mount.h on Release 5 as "return ENOEXEC on exec()". Its bit value is 0x8000. Release 6 does not have anything at that bit value or a function like this at another value. Using this call on Release 6 results in EINVAL.

2.1.7.11 nice(2)

On Release 6, nice can only work on processes in the time sharing class, and returns EINVAL on attempts to change the priority of a fixed priority class process. On Release 5, nice does not return EINVAL. If a Release 5 application is run on Release 6 and does a nice on a fixed priority class process, it would get an unexpected EINVAL error.

To fix, alter the Release 5 application source code to expect EINVAL as a possible error return on using nice, and then recompile.

2.1.7.12 paccess(S)

This Release 5 system call is unsupported by the SVR5 ABI. The **paccess**(S) system call on Release 5 gets and sets parameters in the u area of the current process and is used by a parent process to trace a child process.

An OSR5 ABI application using paccess must be recoded to use the /proc file system to access status information about a process. See **proc**(F).

2.1.7.13 pipe(2)

With the SVR5 ABI, pipes are implemented as two bidirectional, STREAMs-based file descriptors. For the OSR5 ABI, a pipe is implemented as two one-way file descriptors, with one fd being 'send' and the other 'receive'. Release 5 applications run on Release 6 will behave properly despite the difference in implementation.

On Release 6, an **fstat** (see **stat**(S)) of a pipe returns the sum total of bytes available to be read on both pipes as the byte count of the pipe. On Release 5, it is just the data on the inward bound pipe.

The write side of a pipe on Release 5 blocks after 5120 bytes of data. On Release 6, the write side could block considerably later (depending on water marks and system resources). If an OSR5 ABI application depends on the write side blocking after 5120 bytes of data, it will probably not work correctly.

2.1.7.14 poll(2)

The OSR5 ABI poll code shows a system wide tunable `poll_delay_compatibility` that effects whether a poll call with no file descriptors will honor the timeout; if the tunable is set, the timeout is honored.

OSR5 ABI binaries executed on Release 6 will behave the same as on Release 5 (i.e., the `poll_delay_compatibility` tunable is checked and the timeout is honored only if the tunable is set).

2.1.7.15 priocntl(2)

The SVR5 ABI supports a few documented commands not supported by the OSR5 ABI: PC_ADMIN, PC_SETAGEPARMS, and PC_GETAGEPARMS.

2.1.7.16 probe(S)

On Release 5, this undocumented system call executes all the configured probe routines and returns the information they return to the calling user process. This system call is not supported on Release 6. OSR5 ABI applications run on Release 6 will return ENOSYS if they make a call to **probe**.

2.1.7.17 ptrace(2)

An OSR5 ABI application using **ptrace** should be recoded to use the */proc* file system, which is the preferred interface for accessing status information about a process. See **proc**(F).

While **ptrace**(S) is supported on Release 6, only requests ‘0’ through ‘9’ are supported. The Release 5 kernel supports ‘0’ through ‘13’.

To run on Release 6, an OSR5 ABI application that needs to use **ptrace** would probably need to be recompiled. It is highly recommended that such an application be rewritten to use */proc* instead.

2.1.7.18 setcontext(2)

There are differences in the `ucontext` structure used by this system call on Release 6 and Release 5:

- On Release 6, `sigset_t` is 4 `int` fields; there is a `uc_privatedatap long` and 4 `long` fillers at the end of `ucontext_t`.
- On Release 5 `sigset_t` is 1 `int`, but there are 3 `int` fillers. At the end of `ucontext_t` there are 5 `long` fields.

This incompatibility is managed by the Release 6 kernel for OSR5 ABI applications provided these applications always precede a call to **setcontext** with a call to **getcontext**. If an OSR5 ABI application running on Release 6 makes a call to **setcontext** that is not preceded by a call to **getcontext**, the application could pass bad data to **setcontext**, with unpredictable results.

2.1.7.19 sysi86(2)

On Release 5, the SI86GETFEATURES call is used extensively by *libc* and other libraries to determine if a feature is available in the kernel. This subcommand is supported on Release 6, but FEATURE_TCGETS and FEATURE_SID will be turned off, as these services are not provided by the Release 6 kernel.

There are a number of sysi86 subcommands that are on Release 5 but not on Release 6. If an OSR5 ABI executable invokes any of these subcommands, the Release 6 kernel returns EINVAL:

```
0x23 (there is no mnemonic for this command in the sysi86 OSR5 ABI)
SI86KSTR
SI86SWAP
SI86SWPI
RDUBLK
SI86VM86
SI86GETPIPE
SI86SETPIPE
SI86POPPIPE
SI86GETNPIPE
SI86SETPIPE_NM
SI86GETPIPE_ALL
SI86APM
SI86TIMECHG
SANUPD
SI86CAUNICENTER
SI86SETSYSLOG
```

2.1.7.20 sysinfo

The following commands are not supported by the OSR5 ABI:

```
SI_SET_HOSTNAME
SI_HW_PROVIDER
SI_HW_SERIAL
SI_ARCHITECTURE
SI_INITTAB_NAME
SI_BUSTYPES
SI_KERNEL_STAMP
SI_OS_BASE
SI_OS_PROVIDER
SI_USER_LIMIT
```

2.1.7.21 syssetconf

This Release 5 system call and the kernel tunables it provides are unsupported on Release 6.

The Release 5 **syssetconf** system call is an extension to **sysconf** that provides a set of 'kernel' type tunables, none of which are documented on the Release 5 **sysconf**(S) man page (**syssetconf** itself is undocumented). These constants are visible to applications, however, in the Release 5 *sys/unistd.h* header file, so an OSR5 ABI application can use them. None of these values are defined on Release 6. OSR5 ABI applications run on Release 6 that use **syssetconf** on Release 6 will return ENOSYS (and/or a SIGSYS signal). Such applications need to be recoded to work properly on Release 6.

Note that on Release 6, **sysconf**(S) is implemented as a C library function, not a system call.

2.1.7.22 uadmin(2)

There are two commands for this system call on Release 5 that are unsupported on Release 6: A_BDEVSYNC, A_GETDEV, and AD_PWRNAP. OSR5 ABI applications using these commands on Release 6 will get an ENOSYS error return and a SIGSYS signal).

On Release 6, the AD_HALT subcommand (to A_SHUTDOWN) shuts the system down, but does not support the "reboot on keystroke" functionality supported by Release 5.

2.1.7.23 ulimit(2)

The Release 6 system call supports two more commands than Release 5: UL_GMEMLIM and UL_GDESLIM.

2.1.7.24 waitid(2)

The `siginfo_t` structure used by this call differs between Release 5 and Release 6. For OSR5 ABI application binaries running on Release 6, these differences are managed by the kernel. An OSR5 ABI application being ported to the SVR5 ABI on Release 6 might need to be changed to conform to the SVR5 version of `siginfo_t`. Both versions have the same overall size, but there are differences in members and internal offsets, as follows:

- Although both structures use the `pid_t` data type, this type is a 32-bit value on Release 6 and a 16-bit value on Release 5.
- There are differences in `_proc._pdata` for the `_kill` and `_cld` structures within `siginfo_t`.
- The `_fault` structure in Release 6 has additional members not on Release 5.

See the header files */usr/include/sys/siginfo.h* and */osr5/usr/include/sys/siginfo.h* for details.

2.1.7.25 xsetre[gu]id(S)

The Release 5 **xsetregid**(S) and **xsetreuid**(S) functions allow setting the 'real' and 'effective' uids and gids in one system call. There are versions supported on Release 5 in both *libc.so.1* and *lib-socket.a*. OSR5 ABI applications must have been compiled using the version in *libc.so.1* to run on Release 6 (applications linked with static archives are unsupported on Release 6).

When an OSR5 ABI application using these calls is run on Release 6, these calls are mapped to the **setreuid**(S) and **setregid**(S) functions.

2.2 API Compatibility

The API compatibility sections list each interface that is part of the documented APIs defined by the various libraries and header files shipped with SCO OpenServer Release 6 that have substantial differences from the same API on Release 5.

Each section describes the interface compatibility for OSR5 ABI applications as well as compatibility considerations for applications compiled with the SVR5 ABI.

2.2.1 Manual Pages

The same UNIX function can sometimes be in different headers and/or libraries depending on the platform. Use the **man** command to find the location of a function on SCO OpenServer 6:

```
$ man recv

recv(S)

recv -- receive a message from a socket

Synopsis

cc [options] file -lsocket -lnsl
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recv(int socket, void *buf, size_t len,
int flags);
...
```

The top of the man page indicates the headers to use in the source code and the libraries to reference in the makefile. Note that OSR5 API manual pages for libraries are generally available in manual page sections with the suffix "-osr5". For example, the OSR5 ABI Section (S) pages are available on OpenServer 6 in Section (S-osr5).

You can also use DocView (**http://localhost/cgi-bin/manform**) to search for maual pages. If you don't find the page you are looking for on your local OpenServer 6 system, you can also check the manual pages on the web:

- **http://osr600doc.sco.com/en/Navpages/sectionlist.html**

  The complete OpenServer 6 documentation set, with all SVR5 ABI manual pages (and many OSR5 ABI pages).

- **http://osr507doc.sco.com/en/Navpages/sectionlist.html**

  The complete OpenServer 5.0.7 documentation set, with all OSR5 ABI manual pages.

2.2.2 C library (libc) interfaces

The C library (*libc*) contains the system calls and C library routines described in the Section S manual pages.

This "API Compatibility" on page 35 topic is concerned with *libc* function calls; for compatibility information for system calls defined in *libc* (and compatibility for other basic system services), see "Kernel compatibility" on page 23.

C library compatibility with earlier releases is provided through these libraries on Release 6:

/usr/ccs/lib/libc.so	The Release 6 compile-time *libc*. This is the version used (by default) by the Release 6 link editor. It defines the Release 6 *libc* API.
/usr/lib/libc.so.1	The Release 6 run-time libc. This is the version of libc loaded by the dynamic linker by default into every process on the system. It defines the Release 6 *libc* API.
/osr5/usr/lib/libc.so.1	The Release 5 run-time *libc*. For OSR5 ABI applications only. This is the version of *libc* loaded by the dynamic linker into every process on the system that is running a binary marked as a Release 5 binary.

Binary compatibility for OSR5 ABI-compiled binaries is provided through the */osr5/usr/lib/libc.so.1* version of *libc*, which is loaded by the dynamic linker for any executable that is appropriately marked as an OSR5 binary (see **elfmark**(CP).

2.2.2.1 tm structure

The `tm` structure is used by various time functions in *libc* (**asctime, asctime_r, ascftime, getdate, gmtime, gmtime_r, localtime, localtime_r, mktime, strftime, strptime, wcsftime**).

The `tm` structure on Release 5 has two elements at the end of the structure that are not present on Release 6:

```
long tm_tzadj;        /* seconds from UTC (east < 0) */
char tm_name[LTZNMAX]; /* name of timezone */
```

On Release 6, these values are available only in the external variables `timezone` and `tzname`, respectively; these external variables are also available on Release 5. Portable applications should depend only on the external variables, not on the structure elements.

2.2.2.2 confstr

The following values for the *name* argument to **confstr** (defined in *unistd.h*) are valid on Release 6 but not on Release 5:

_CS_HOSTNAME	2

_CS_RELEASE	3
_CS_VERSION	4
_CS_MACHINE	5
_CS_ARCHITECTURE	6
_CS_HW_SERIAL	7
_CS_HW_PROVIDER	8
_CS_SRPC_DOMAIN	9
_CS_INITTAB_NAME	10
_CS_SYSNAME	11

2.2.2.3 dlsym

The **RTLD_NEXT** argument for the **handle** parameter is unsupported on Release 5.

2.2.2.4 fnmatch

The Release 6 and Release 5 implementations of **fnmatch** differ in the header file declarations of flags and return values found in *fnmatch.h*. Basically, the Release 6 interface is a superset of Release 5. Some constants are defined with different values; these are managed by the Release 6 kernel.

The Release 6 interface supports these additional flags:

```
#define FNM_BADRANGE 0x008/* accept [m-a] ranges as [ma] */
#define FNM_BKTESCAPE 0x010/* allow in []s to quote next anything */
#define FNM_EXTENDED 0x020 /* use full ksh-style patterns */
#define FNM_RETMIN 0x040/* return length of minimum match */
#define FNM_RETMAX 0x080/* return length of maximum match */
#define FNM_REUSE 0x100 /* reusing this FNM */
#define FNM_COMPONENT 0x200 /* only matching a component */
#define FNM_SUBEXPR 0x400 /* fill fnm_nsub, fnm_so[], fnm_eo[] */
#define FNM_UNANCHORED 0x800 /* match need not include string start */
#define FNM_NSUBEXPR 10 /* length of fnm_so[] and fnm_eo[] */
```

Also, the Release 6 interface supports one additional error:

```
#define FNM_ERROR (-4) /* internal error; probably allocation failure */
```

A portable application should use only those flags common to the two systems, or first determine the system on which it is running and then use flags appropriate to that system.

2.2.2.5 ftw/nftw

The constants used as argument values for ftw and nftw (and defined in ftw.h) on Release 5 and

Release 6 are slightly different. A portable program should only use those values common to the two systems or first determine the system on which it is running before issuing the call with appropriate values.

The Release 6 values are a superset of the Release 5 values; the extensions provided are:

```
#define FTW_ERR 8 /* FTW_ERRORS only; internal nftw() failure */
#define _FTW_FINDDIR 020 /* continue even if getcwd() fails */
#define FTW_TRYCHDIR (FTW_CHDIR|_FTW_FINDDIR) /* for nftw() to try getcwd
*/
#define FTW_ERRORS 040 /* call fcn for nftw() internal errors, too */
```

2.2.2.6 glob/globfree

Release 6 and Release 5 have different constant and structure definitions in *glob.h* for the **glob**(S) routine.

While the `glob_t` structure has elements on both systems not found on the other, these elements are for system use only, and should not affect application compatibility.

```
/* SVR5 glob_t structure */
typedef struct {
struct gl_str *gl_str; /* for memory management */
char **gl_pathv; /* list of matched pathnames */
size_t gl_pathc; /* length of gl_pathv[] (less 1) */
size_t gl_offs; /* slots to reserve in gl_pathv[] */
} glob_t;

/* OSR5 glob_t structure */
typedef struct {
size_t gl_pathc; /* count of paths matched by pattern */
char **gl_pathv; /* pointer to list of matched pathnames */
size_t gl_offs; /* slots to reserve in gl_pathv[] */
/* Internal SCO variables */
size_t gl_vnum; /* Number of entries in gl_pathv */
size_t gl_vmax; /* Maximum entries in gl_pathv */
} glob_t;
```

The manifest constants supported on Release 6 are a superset of those provided on Release 5.

The additional constants are:

```
#define GLOB_FULLMARK 0x0080 /* append “/”, “@”, “*”, “|” like ls(1) */
#define GLOB_NOCOLLATE 0x0100 /* use “C” sorting order */
#define GLOB_OKAYDOT 0x0200 /* permit leading . to match specials */
#define GLOB_BADRANGE 0x0400 /* accept [m-a] ranges as [ma] */
#define GLOB_BKTESCAPE 0x0800 /* allow in []s to quote next anything */
#define GLOB_EXTENDED 0x1000 /* use full ksh-style patterns */
```

Some constants are defined with different values; these are managed by the Release 6 kernel for OSR5 ABI applications.

2.2.2.7 iconv

The implementation of **iconv** on Release 5 supports a flag mechanism that is not supported by Release 6 (or by the X/Open standard). This mechanism allows you to set a flag in a state field in the conversion descriptor used as input to iconv.

These flags (_ICONV_DELETE and _ICONV_COMP) and the state field in the conversion descriptor are not supported by Release 6. OSR5 ABI binaries that run on Release 6 will fail if they attempt to set the conversion descriptor state field with these flags.

2.2.2.8 isnan/isnand/isnanf

On Release 5, it is necessary to include *nan.h*, *math.h*, and *ieeefp.h* to compile a program using these functions. On Release 6, *nan.h* does not exist, so include only *math.h* and *ieeefp.h*.

2.2.2.9 jmp_buf

On Release 5, the size of the `jmp_buf` used by **longjmp** and **setjmp** is defined in *setjmp.h* as 6; on Release 6, it is 10. This presents no problem for source code portability. OSR5 ABI binaries that use `jmp_buf` will run correctly on Release 6.

2.2.2.10 mallinfo

The Release 5 implementation of **mallinfo** is available only in *libc.a*.

2.2.2.11 nl_langinfo

The header file *langinfo.h* on Release 6 contains a larger set of manifest constants than the same header file on Release 5. A portable application should only expect the constants that are defined on both systems to be available.

The additional constants defined on Release 6 in langinfo.h are:

```
#define _MAXSTRMSG 57 /* Maximum number of strings in langinfo */
#define QUITSTR 58 /* “yes, go away” answer */
#define QUITEXPR 59 /* “go away” response ERE string */
#define DATECMD_FMT 60 /* format string used by date(1) */
#define CHARCLASS61 /* all valid wctype() strings, ;-separated */
```

Also, some constants are defined with different values; these are managed by the Release 6 kernel for OSR5 ABI applications.

2.2.2.12 passwd structure

Several routines make use of the `passwd` structure defined in *pwd.h*. This structure contains two members (`pw_uid` and `pw_gid`) whose types (`uid_t` and `gid_t`, respectively) are defined differently depending on the operating system.

On Release 6, `uid_t` and `gid_t` are both defined as `long`; on Release 5, they are both `unsigned short`. The Release 5 implementation makes up for this difference by adding two pad fields to the passwd structure, as shown below:

```
uid_t pw_uid;/* user ID */
short __pad1;/* padded space */
gid_t pw_gid;/* group ID */
short __pad2;/* padded space */
```

2.2.2.13 sigset_t

On Release 5, the `sigset_t` data type (use by the functions **sigaddset**, **sigdelset**, **sigemptyset**, **sigfillset**, and **sigismember**) is declared a `long` in *sys/signal.h*. On Release 6, it is declared as follows:

```
typedef struct {  /* signal set type */
unsigned int sa_sigbits[4];
} sigset_t;
```

The Release 6 kernel will put data from an OSR5 ABI binary into the Release 6 structure before executing the function.

Also see “setcontext(2)” on page 32.

2.2.2.14 sysconf

The following Release 5 **sysconf** commands are not supported on Release 6:

```
_SC_32BIT_INODES
_SC_FSYNC
_SC_KERNEL_PROC
_SC_KERNEL_PROC_MAX
_SC_KERNEL_REGION
_SC_KERNEL_REGION_MAX
_SC_KERNEL_FILE
_SC_KERNEL_FILE_MAX
_SC_KERNEL_INODE
_SC_KERNEL_INODE_MAX
_SC_KERNEL_S5INODE
_SC_KERNEL_S5INODE_MAX
_SC_KERNEL_DISK
_SC_KERNEL_DISK_MAX
_SC_KERNEL_CLIST
_SC_KERNEL_CLIST_MAX
_SC_KERNEL_DMABUF
_SC_KERNEL_DMABUF_MAX
_SC_KERNEL_MOUNT
_SC_KERNEL_MOUNT_MAX
_SC_KERNEL_FLCKREC
_SC_KERNEL_FLCKREC_MAX
_SC_KERNEL_PINODE
_SC_KERNEL_PINODE_MAX
_SC_MAPPED_FILES
```

2.2.2.15 ttyslot

The file accessed by **ttyslot** on Release 5 is */var/utmp*, while on Release 6 it is */var/adm/utmp*. For OSR5 ABI binaries on Release 6, the kernel directs the program to the proper file.

2.2.3 BSD library (libucb) interface

The BSD Compatibility library (*libucb*) is present on SCO OpenServer 6, but it is usually not needed since many of the functions in this library are now in the native Release 6 libraries as well.

2.2.4 Threads and asynchronous I/O (libthread) interfaces

POSIX threads (*libpthread*) are documented on the Section PTHREAD manual pages.

The SVR5 threads library interfaces (*libthread*), which include the asynchronous I/O routines, are documented in the Section THREAD manual pages and Section AIO manual pages. Note that the asynchronous I/O routines on Release 5 that correspond to the routines documented in the Release 6 Section AIO manual pages, are part of *libsuds* on Release 5, and are not supported for OSR5 ABI binaries running on Release 6. The *libsuds* library and its versions of these routines are also not supported by the SVR5 ABI.

2.2.5 Network support library (libnsl) interfaces

2.2.5.1 AUTH structure

The AUTH structure (see */usr/include/rpc/auth.h*) has an extra element (a `des_block` structure) that is not in the Release 5 implementation.

2.2.5.2 CLIENT structure

The CLIENT structure (see */usr/include/rpc/clnt.h*) has elements not in the Release 5 implementation.

2.2.5.3 Berkeley style client calls

These calls (**callrpc**, **clnttcp_create**, **clntudp_create**, **clntudp_bufcreate**, and **clntraw_create**) are present in *libnsl* on Release 6 for compatibility only and are undocumented. To compile a program that uses them, you must include the */usr/include/rpc/clnt_soc.h* header file. On Release 6 these client interfaces are replaced by the **rpc_clnt_create**(S) interfaces.

These calls also access the CLIENT structure; see above.

2.2.5.4 Portmapper

These interfaces (**pmap_set**, **pmap_unset**, **pmap_rmtcall**, **pmap_getmaps**, **pmap_getport**, and **clnt_broadcast**) are present in *libnsl* on Release 6 for compatibility only and are undocumented. To compile a program that uses them, you must include the */usr/include/rpc/pmap_clnt.h* header file. On

Release 6, the **rpcbind**(S) interfaces replace the portmapper interfaces.

2.2.5.5 Berkeley style service calls

These calls (**svcfd_create**, **svcraw_create**, **svctcp_create**, **svcudp_bufcreate**, **svcudp_create**, and **svcudp_enablecache**) are present in *libnsl* on Release 6 for compatibility only and are undocumented. To compile a program that uses them, you must include the */usr/include/rpc/svc_soc.h* header file. On Release 6 these client interfaces are replaced by the **rpc_svc_create**(S) interfaces.

These calls also access the SVCXPRT structure; see below

2.2.5.6 SVCXPRT structure

The SVCXPRT structure (see */usr/include/rpc/svc.h*) has elements not in the Release 5 implementation. This affects the folowing routines: **svcerr_auth**, **svcerr_decode**, **svcerr_noproc**, **svcerr_noprog**, **svcerr_progvers**, **svcerr_systemerr**, and **svcerr_weakauth**.

2.2.6 Transport interface (XTI and TLI)

SCO OpenServer Release 5 and Release 6 both support the X/Open Transport Interface (XTI) and the Transport Level Interface (TLI). Both XTI and TLI are APIs that allow user processes to access transport providers in a (mostly) transport-independent fashion.

The Transport Level Interface was modeled after the industry standard ISO Transport Service Definition (ISO 8072). The resulting interface was called the Transport Level Interface (TLI) library, first introduced with AT&T UNIX System V Release 3.0 in 1986.

XTI is the X/Open-sponsored successor to TLI, and is defined in the Networking Services, Issue 5 document.

From the standpoint of syntax and semantics, the two libraries are nearly identical, but XTI is the preferred interface for new applications, since it is the industry standard. The TLI library routines are maintained for compatibility with previous releases only.

The Release 6 transport level functions are defined in *libnsl* and support both the older TLI semantics and the newer XTI semantics. The library entry points have their traditional names for the TLI functions, such as `t_open`. For the XTI functions, however, the entry points have new names, such as `_xti_open`. When applications are compiled with the SVR5 ABI, the TLI function names are translated to the XTI entry points by macros by including the file *xti.h* in the source code.

In this way, Release 6 is able to support older applications that use the TLI interface, while still providing the new XTI interface.

To compile and link a program using XTI semantics, do the following:

1. Include the *xti.h* header file at the beginning of your source files. The syntax of the include preprocessor directive to use is:

   ```
   #include <xti.h>
   ```

2. Specify the *libnsl* library as one of the libraries to be searched on the **cc** command line:

```
cc option file -lnsl
```

To compile a program using the TLI interfaces exclusively, include the TLI header file (*tiuser.h*) instead of the XTI header file. The syntax of the include preprocessor directive to use is:

```
#include <tiuser.h>
```

2.2.7 Sockets interface

SCO OpenServer Release 5 and Release 6 support the sockets interface. Although there are many similarities between the implementations, the differences are significant enough that a compatibility library is provided in */osr5/usr/lib* so that OSR5 ABI binaries can run on Release 6 without modification.

The default action on Release 6 is UNIX95 sockets behavior; this essentially translates to linking with */usr/lib/libsocket.so.2*.

To compile and link a program using SVR5 sockets, do the following:

1. Include the following header files at the beginning of your source files. The syntax of the include preprocessor directives to use are:

```
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
```

2. Specify the *libsocket* and *libnsl* libraries as two of the libraries to be searched on the **cc** command line, in the order shown:

```
cc -lsocket -lnsl ...
```

or use:

```
cc -lxnet ...
```

2.2.7.1 Socket addressing

The type of socket address structure passed to a socket routine depends on the address and protocol families used by your application.

SVR5	AF_UNIX AF_INET AF_INET6	AF_UNIX requires a `socketaddr_un` structure as defined in *sys/un.h*. AF_INET requires a `sockaddr_in` structure as defined in *netinet/in.h*. AF_INET6 requires a `sockaddr_in6` structure as defined in *netinet/in6.h*. Each of these Release 6 `sockaddr` structures have been modified to support variable length sockets. The net result of this modification is that the `family` member has been shortened to 8 bits and a new 8-bit member inserted before it called `len`.
OSR5	AF_UNIX AF_INET	AF_UNIX requires a `socketaddr_un` structure as defined in *sys/un.h*. AF_INET requires a `sockaddr_in` structure as defined in *netinet/in.h*. Variable length sockets are not supported.

2.2.7.2 accept

In Release 5, the `*addrlen` parameter is an `int`. In Release 6, it is declared as `size_t`.

Release 6 can return these additional errors (which are not returned by Release 5):

EWOULDBLOCK	The socket is marked as non-blocking and no connections are present to be accepted.
EPROTO	A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.
ENODEV	The protocol family and type corresponding to s could not be found in the netconfig file.
ENOMEM	There was insufficient user memory available to complete the operation.
ENOSR	There were insufficient STREAMS resources available to complete the operation.

Release 5 can return EFAULT (the addr parameter is not in a writable part of the user address space), which is not returned by Release 6.

2.2.7.3 bind

In Release 5, the `*namelen` parameter is an `int`. In Release 6, it is declared as `size_t`.

Release 6 can return these additional errors (which are not returned by Release 5):

EINVAL	namelen is not the size of a valid address for the specified address family.
ENOSR	There were insufficient STREAMS resources for the operation to complete.

EAFNOSUPPORT	The specified address is not a valid address for the address family of the specified socket.
EOPNOTSUPP	The socket type of the specified socket does not support binding to an address.
EISCONN	The socket is already connected.
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
ENOBUFS	Insufficient resources were available to complete the call.
ENOTDIR	A component of the path prefix of the pathname in name is not a directory.
ENOENT	A component of the path prefix of the pathname in name does not exist, or the pathname is an empty string.
EACCES	Search permission is denied for a component of the path prefix of the pathname in name, or the requested name requires writing in a directory with a mode that denies write permission.
ELOOP	Too many symbolic links were encountered in translating the pathname in name.
ENAMETOOLONG	A component of the pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX characters.
EIO	An I/O error occurred while making the directory entry or allocating the inode.
EROFS	The inode would reside on a read-only file system.
EISDIR	A null pathname was specified.

Release 5 can return EFAULT (the `name` parameter is not in a valid part of the user address space), which is not returned by Release 6.

2.2.7.4 connect

In Release 5, the third parameter is called `*namelen` and is an `int`. In Release 6, it is called `*address_len` and is declared as `size_t`. The parameters have the same use (i.e., they contain the length of the previous calling parameter).

Release 6 can return these additional errors (which are not returned by Release 5):

EALREADY	A connection request is already in progress for the specified socket.

EINPROGRESS	The socket is non-blocking and the connection cannot be completed immediately; the connection will be established asynchronously.
EINTR	The connection attempt was interrupted before any data arrived by the delivery of a signal. The connection will be established asynchronously.
EPROTOTYPE	The file referred to by address is a socket of a type other than the socket bound to the specified peer address.
ECONNRESET	The remote host reset the connection request.
EINVAL	address_len is not the size of a valid address for the specified address family, or invalid address family in sockaddr structure.
ENAME-TOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
ENETDOWN	The local interface used to reach the destination is down.
ENOBUFS	No buffer space is available.
ENOSR	There were insufficient STREAMS resources available to complete the operation.
ENOTDIR	A component of the path prefix of the pathname in address is not a directory.
ENAME-TOOLONG	A component of a pathname exceeded NAME_MAX characters or an entire pathname exceeded PATH_MAX.
EACCES	Search permission is denied for a component of the path prefix or write access to the named socket is denied.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating the pathname in address.
ENOENT	A component of the pathname does not name an existing file or the pathname is an empty string.

2.2.7.5 ether_aton

This returns an `ether_addr_t` (unsigned char) on Release 6, while it returns an `ether_addr` structure (which has one member of type `ether_addr_t`) on Release 5.

2.2.7.6 ether_hostton

This function takes an `ether_addr_t` (unsigned char) parameter on Release 6, while it takes an `ether_addr` structure (which has one member of type `ether_addr_t`) on Release 5.

2.2.7.7 ether_line

This function takes an `ether_addr_t` (unsigned char) parameter on Release 6, while it takes an `ether_addr` structure (which has one member of type `ether_addr_t`) on Release 5.

2.2.7.8 ether_ntoa

This function takes an `ether_addr_t` (unsigned char) parameter on Release 6, while it takes an `ether_addr` structure (which has one member of type `ether_addr_t`) on Release 5.

2.2.7.9 ether_ntohost

This function takes an `ether_addr_t` (unsigned char) parameter on Release 6, while it takes an `ether_addr` structure (which has one member of type `ether_addr_t`) on Release 5.

2.2.7.10 ftruncate/truncate

These functions appear as system calls in libc on Release 6 and Release 5; Release 5 also provides versions of these in the sockets library.

2.2.7.11 getpeername/setpeername

In Release 5, the *`namelen` parameter is an `int`. In Release 6, it is declared as `size_t`.

Release 6 can return these additional errors (which are not returned by Release 5):

ENOMEM	There was insufficient user memory for the operation to complete.
ENOSR	There were insufficient STREAMS resources for the operation to complete.
EINVAL	The socket has been shut down.

2.2.7.12 getsockname/setsockname

In Release 5, the *`namelen` parameter is an `int`. In Release 6, it is declared as `size_t`.

Release 6 can return these additional errors (which are not returned by Release 5):

EOPNOTSUPP	The operation is not supported for this socket's protocol.
ENOMEM	There was insufficient user memory for the operation to complete.
ENOSR	There were insufficient STREAMS resources for the operation to complete.
EINVAL	The socket has been shut down.

2.2.7.13 getsockopt/setsockopt

In Release 5, the *`optlen` parameter is an `int`. In Release 6, it is declared as `size_t`.

Release 6 can return these additional errors (which are not returned by Release 5):

ENOMEM	There was insufficient user memory for the operation to complete.
ENOSR	There were insufficient STREAMS resources for the operation to complete.

The following options are provided on Release 5, but are not supported on Release 6:

SO_REUSEPORT	toggle on/off local port reuse
SO_SNDLOWAT	set low-water mark for output
SO_RCVLOWAT	set low-water mark for input
SO_PROTOTYPE	get/set the protocol number associated with the stream

2.2.7.14 gettimeofday/settimeofday

These routines are implemented as system calls in Release 6. On Release 5, they are implemented as routines in the *libsocket* library, as well as system calls (in *libc*).

The syntax of the system call on Release 6 is:

```
#include <sys/time.h>
int gettimeofday(struct timeval *tp, void *reserved);
int settimeofday(struct timeval *tp, void *reserved);
```

where the second parameter is required to be NULL.

On Release 5, the system call syntax is:

```
#include <sys/time.h>
int gettimeofday(struct timeval *tp);
int settimeofday(struct timeval *tp);
```

The Release 5 *libsocket* routine's syntax is:

```
#include <sys/time.h>
int gettimeofday(struct timeval *tp , struct timezone *tpz);
int settimeofday(struct timeval *tp , struct timezone *tpz);
```

The `timeval` structures used are the same on each platform; but the `timezone` structure is unique to the Release 5 *libsocket* routines (and any data it contains is ignored on the other platforms):

```
struct timezone {
int tz_minuteswest; /* of Greewich */
```

```
int tz_dsttime; /* type of dst correction to apply */
};
```

2.2.7.15 listen

Release 6 can return these additional errors (which are not returned by Release 5):

EINVAL	The socket is already connected or has been shut down.
EDESTADDRREQ	The socket is not bound to a local address, and the protocol does not support listening on an unbound socket.
ENOBUFS	System resources are insufficient to complete the call.

2.2.7.16 netgroup

The **netgroup** routines (**endnetgrent**, **getnegrent**, **setnetgrent**, **innetgr**) on Release 5 require that the Network Information Service (NIS) is running for these routines to return successfully.

This is not a requirement on Release 5, on which these routines also check the file */etc/netgroup* for network group information.

2.2.7.17 recv/recvfrom/recvmsg

1. The **recvmsg** call uses a `msghdr` structure to minimize the number of directly supplied parameters. The Release 6 structure is defined in *sys/socket.h* and includes the following members:

   ```
   void * msg_name;        /* optional address */
   size_t msg_namelen;     /* size of address */
   struct iovec * msg_iov; /* scatter/gather array */
   int    msg_iovlen;      /* number of elements in msg_iov */
   void * msg_accrights;   /* access rights sent/received */
   int    msg_accrightslen;
   void * msg_control;
   size_t msgcontrollen;
   int    msg_flags;
   ```

   In Release 5, this structure is defined as follows:

   ```
   caddr_t msg_name;               /* optional address */
   int     msg_namelen;            /* size of address */
   struct iovec    *msg_iov;       /* scatter/gather array */
   int     msg_iovlen;             /* # elements in msg_iov */
   caddr_t msg_control;            /* control information sent/received */
   int     msg_controllen;         /* size of control information */
   int     msg_flags;              /* size of control information */
   ```

2. The use of the `cmsghdr` structure for control data along with the CMSG macros are supported in Release 5 and Release 6. In Release 5 the `cmsghdr` structure is defined as:

```
int cmsg_level; /* originating protocol */
int cmsg_type; /* protocol-specific type */
u_int cmsg_len; /* data byte count, including hdr */
```

In Release 6 the `cmsghdr` structure is defined as:

```
size_t cmsg_len; /* data byte count, including hdr */
int cmsg_level; /* originating protocol */
int cmsg_type; /* protocol-specific type */
```

3. Release 6 can return these additional errors (not returned on Release 5):

ECONNRESET	A connection was forcibly closed by a peer.
ENOTSOCK	socket is a descriptor for a file, not a socket.
EINTR	The operation was interrupted by delivery of a signal before any data was available to be received.
EINVAL	MSG_OOB is set and there is no available out-of-band data.
ENOTCONN	A receive is attempted on a connection-oriented socket that is not connected.
EWOULDBLOCK	The socket is marked non-blocking and the requested operation would block.
EOPNOTSUPP	The specified flags are not supported for this socket type or protocol.
ETIMEDOUT	The connection timed out during connection or because of a transmission timeout on active connection.
EIO	An I/O error occurred while reading to or writing from the file system.
ENOBUFS	System resources were insufficient to perform the operation.
ENOMEM	There was insufficient user memory available for the operation to complete.
ENOSR	There were insufficient STREAMS resources for the operation to complete.

2.2.7.18 select

On Release 5, the FD_SETSIZE constant is normally defined in *sys/types.h* as either 150 or 11000; on Release 6, the value of FD_SETSIZE is 4096. See **select**(S).

2.2.7.19 send/sendto

In Release 5, the `*len` and `*tolen` meters are `int values`. In Release 6, they are declared as `size_t`.

Release 6 can return these additional errors (which are not returned by Release 5):

EINVAL	`tolen` is not the size of a valid address for the specified address family.

EINTR	The operation was interrupted by delivery of a signal before any data could be buffered to be sent.
EWOULDBLOCK	The socket is marked non-blocking and the requested operation would block.
ENOMEM	There was insufficient user memory available for the operation to complete.
ENOSR	There were insufficient STREAMS resources available for the operation to complete.

See ‘‘recv/recvfrom/recvmsg’’ for a description of platform differences in the `msghdr` structure.

2.2.7.20 shutdown

Release 6 can return these additional errors (which are not returned by Release 5):

ENOMEM	There was insufficient user memory available for the operation to complete.
ENOBUFS	System resources were insufficient to perform the operation.
ENOSR	There were insufficient STREAMS resources available for the operation to complete.

2.2.7.21 socket

Release 6 can return these additional errors (which are not returned by Release 5):

ENOMEM	Insufficient user memory is available.
EPROTONOSUPPORT	Protocol not supported.
EMFILE	The system file table is full. (Release 5 returns ENFILE instead.)

Release 5 can return these additional errors (which are not returned by Release 6):

ENFILE	The system file table is full. (Release 6 returns EMFILE instead.)

The protocol families and types differ between the systems. On Release 6, the AF_INET6 and PF_KEY protocol families are supported, in addition to the AF_UNIX and AF_INET protocol families supported on Release 5.

Release 6 supports the SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, SOCK_SEQPACKET, and SOCK_RDM socket types; Release 5 supports only SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW.

2.2.7.22 openlog

On Release 5, */usr/include/sys/syslog.h* contains the following facilities definitions:

```
#define LOG_CRON (9<<3) /* clock daemon */
#define LOG_AUTHPRIV (10<<3) /* security/authorization msgs (private) */
```

On Release 6, the above lines are omitted, and the following appear instead:

```
#define LOG_LFMT (14<<3) /* logalert facility */
#define LOG_CRON (15<<3) /* cron/at subsystem */
```

2.2.8 Name resolution (libresolv) library routines

The **resolver** library routines in Release 6 are based on the BIND 9 distribution. For OSR5 ABI applications there are no compatibility impacts beyond the relocation of the routines from libsocket to the libresolv library, which is a concern for source code compilation on Release 6 only.

There are differences in resolver options and the _res data structure that holds them that span the use of most of these routines. These impact source compatibility for SCO OpenServer Release 5 applications being recoded to use the SVR5 ABI.

2.2.9 File transfer protocol (ftp) interface

SCO OpenServer Release 5 and Release 6 support the Internet file transfer protocol (FTP) interface developed for SCO OpenServer; this interface is defined in */usr/lib/libftp.so*. Source code that uses this interface will compile on Release 6 without any changes.

To compile and link a program that uses the FTP interface, do the following:

1. Include the *libftp.h* header file at the beginning of your source files. The syntax of the include preprocessor directive to use is:

   ```
   #include <net/libftp.h>
   ```

2. Specify the *libftp* library as one of the libraries to be searched on the **cc** command line:

   ```
   cc options file -lftp
   ```

2.2.10 STREAMS interface

The STREAMS interface on SCO OpenServer Release 5 and Release 6 are fundamentally the same, with the major differences in the stream head ioctl commands.

On all systems, STREAMS the interface is implemented through a set of system calls and ioctl commands (issued using the ioctl system call).

Note that applications compiled with the SCO OpenServer Development System must define _SVID3 in the source in order to use the following **ioctl** commands:

```
I_ATMARK
```

```
I_CANPUT
I_CKBAND
I_FLUSHBAND
I_GETBAND
I_GETCLTIME
I_GWROPT
I_LIST
I_PLINK
I_PUNLINK
I_SETCLTIME
I_SWROPT
```

2.2.10.1 I_GETCLTIME

This **ioctl** returns the close time delay in a `long` on Release 6; on Release 5 it is returned in an `int`.

2.2.10.2 I_RECVFD

The `uid` and `gid` members of the `strrecvfd` structure are defined as `uid_t` and `gid_t` (that is, `long`) on Release 6; on Release 5 they are both defined as an `unsigned short`.

2.2.10.3 I_S_RECVFD

This **ioctl** command is not supported on Release 5.

2.2.10.4 I_SETSIG

For the S_BANDBURG event subcommand, Release 6 returns SIGURG, while Release 5 returns SIGUSR1.

2.2.11 Event queue (libevent) interface

The Release 6 event library (*libevent.so*) is a direct port of the SCO OpenServer Release 5 event API, which allows applications to obtain device events directly.

Essentially, these routines allow a program to manage device events through an event queue. Devices such as mice or keyboards may be read through an event queue. For more information, see the manual pages for the **ev_*** (**ev_block**, **ev_init**, etc.) routines in Section S-osr5.

The event interface is not part of any current industry standard.

2.2.12 SNMP (libsnmp) interface

The version of *libsnmp* supported on Release 6 implements SNMP Version 1 (SNMPv1).

The Release 5 version of *libsnmp* (included in */osr5/usr/lib*) is supported on Release 6 for Release 5 binaries only. By default, the Release 5 library implements SNMP Version 2 (SNMPv2), unless the application is compiled with either SNMPV1 or SNMPV1_ONLY set.

The header files for SNMP are found in */usr/include/snmp* on Release 5, and in */usr/include/netmgt* on Release 6.

Source compatibility for every one of these calls is affected by differences in the data structures used by the SCO OpenServer Release 5 and Release 6 implementations. See the following sections for details:

• ‘‘Object identifier (OID) structure’’

• ‘‘Object type (OT) structure’’

• ‘‘SCO OpenServer 64 bit counters’’

• ‘‘Aggregate structure’’

2.2.12.1 make_varbind

The implementation of **make_varbind** on Release 5 is as follows:

```
VarBindList make_varbind(OID oid_ptr, short type, unsigned long ul_value,
long sl_value, OctetString os_value, OID oid_value, Counter64 *c64_value);
```

The `Counter64` data type as well as the `*c64_value` parameter to this function are not supported on Release 6.

There are differences in `VarBindList` types and error returns between SCO OpenServer Release 6 and Release 5 that impact source compatibility. The Release 5 definitions in *snmp.h* are:

```
/* Universal's */
#define INTEGER_TYPE 0x02
#define GET_REQUEST_TYPE 0xA0
#define GET_NEXT_REQUEST_TYPE 0xA1
#define GET_RESPONSE_TYPE 0xA2
#define SET_REQUEST_TYPE 0xA3
#define TRAP_TYPE 0xA4 /* obsolete */
#define GET_BULK_REQUEST_TYPE 0xA5
#define INFORM_REQUEST_TYPE 0xA6
#define V2_TRAP_TYPE 0xA7
#define BITSTRING_TYPE 0x03
#define OCTET_PRIM_TYPE 0x04
#define DisplayString OCTET_PRIM_TYPE
#define NULL_TYPE 0x05
#define OBJECT_ID_TYPE 0x06
#define OCTET_CONSTRUCT_TYPE 0x24
#define SEQUENCE_TYPE 0x30
#define Aggregate 0xFF
/* Primitive context's */
#define NO_SUCH_OBJECT 0x80
#define NO_SUCH_INSTANCE 0x81
#define END_OF_MIB_VIEW 0x82
/* Application's */
#define IP_ADDR_PRIM_TYPE 0x40
#define COUNTER_TYPE 0x41
```

```
#define GAUGE_TYPE 0x42
#define TIME_TICKS_TYPE 0x43
#define OPAQUE_PRIM_TYPE 0x44
#define IP_ADDR_CONSTRUCT_TYPE 0x60
#define OPAQUE_CONSTRUCT_TYPE 0x64
#define NSAP_ADDR_TYPE 0x45
#define COUNTER64_TYPE 0x46
#define UINTEGER_TYPE 0x47
/* SNMPv2 message related types */
#define PRIV_MSG_TYPE 0xA1
#define PRIV_DATA_TYPE 0x81
#define AUTH_MSG_TYPE 0xA1
#define MD5_AUTH_INFO_TYPE 0xA2
#define NO_AUTH_INFO_TYPE 0x04
#define MGMT_COM_TYPE 0xA2
/* Application's SMUX */
#define SMUX__PDUs_simple 0x60
#define SMUX__PDUs_close 0x41
#define SMUX__PDUs_registerRequest 0x62
#define SMUX__PDUs_registerResponse 0x43
#define SMUX__PDUs_get__request 0xA0
#define SMUX__PDUs_get__next__request 0xA1
#define SMUX__PDUs_get__response 0xA2
#define SMUX__PDUs_set__request 0xA3
#define SMUX__PDUs_trap 0xA4
#define SMUX__PDUs_commitOrRollback 0x44
/* Error codes */
#define NO_ERROR 0
#define TOO_BIG_ERROR 1
#define NO_SUCH_NAME_ERROR 2
#define BAD_VALUE_ERROR 3
#define READ_ONLY_ERROR 4
#define GEN_ERROR 5
#define NO_ACCESS 6
#define WRONG_TYPE 7
#define WRONG_LENGTH 8
#define WRONG_ENCODING 9
#define WRONG_VALUE 10
#define NO_CREATION 11
#define INCONSISTENT_VALUE 12
#define RESOURCE_UNAVAILABLE 13
#define COMMIT_FAILED 14
#define UNDO_FAILED 15
#define AUTHORIZATION_ERROR 16
#define NOT_WRITEABLE 17
#define INCONSISTENT_NAME 18
```

The Release 6 definitions (from */usr/include/netmgt/snmp.h*) are:

```
/* Universal's */
#define INTEGER_TYPE 0x02
#define OCTET_PRIM_TYPE 0x04
```

```
#define DisplayString OCTET_PRIM_TYPE
#define NULL_TYPE 0x05
#define OBJECT_ID_TYPE 0x06
#define OCTET_CONSTRUCT_TYPE 0x24
#define SEQUENCE_TYPE 0x30
#define Aggregate 0xFF
/* Application's */
#define IP_ADDR_PRIM_TYPE 0x40
#define COUNTER_TYPE 0x41
#define GAUGE_TYPE 0x42
#define TIME_TICKS_TYPE 0x43
#define OPAQUE_PRIM_TYPE 0x44
#define IP_ADDR_CONSTRUCT_TYPE 0x60
#define OPAQUE_CONSTRUCT_TYPE 0x64
/* Context's */
#define GET_REQUEST_TYPE 0xA0
#define GET_NEXT_REQUEST_TYPE 0xA1
#define GET_RESPONSE_TYPE 0xA2
#define SET_REQUEST_TYPE 0xA3
#define TRAP_TYPE 0xA4
/* Application's SMUX */
#define SMUX__PDUs_simple 0x60
#define SMUX__PDUs_close 0x41
#define SMUX__PDUs_registerRequest 0x62
#define SMUX__PDUs_registerResponse 0x43
#define SMUX__PDUs_get__request 0xA0
#define SMUX__PDUs_get__next__request 0xA1
#define SMUX_GET_REQUEST_TYPE GET_REQUEST_TYPE
#define SMUX_GET_NEXT_REQUEST_TYPE GET_NEXT_REQUEST_TYPE
#define SMUX_GET_RESPONSE_TYPE GET_RESPONSE_TYPE
#define SMUX_SET_REQUEST_TYPE SET_REQUEST_TYPE
#define SMUX_TRAP_TYPE TRAP_TYPE
#define SMUX__PDUs_get__response 0xA2
#define SMUX__PDUs_set__request 0xA3
#define SMUX__PDUs_trap 0xA4
#define SMUX__PDUs_commitOrRollback 0x44
#define SMUX_SIMPLE_TYPE 0x60
#define SMUX_CLOSE_TYPE 0x41
#define SMUX_REG_REQUEST_TYPE 0x62
#define SMUX_REG_RESPONSE_TYPE 0x43
#define SMUX_SOUT_TYPE 0x44
/* Error codes */
#define NO_ERROR 0
#define TOO_BIG_ERROR 1
#define NO_SUCH_NAME_ERROR 2
#define BAD_VALUE_ERROR 3
#define READ_ONLY_ERROR 4
#define GEN_ERROR 5
```

2.2.12.2 parse_pdu

On Release 5, the **parse_pdu** routine is passed a packet and length argument, as follows:

```
Pdu *parse_pdu(u_char **packet, long *length)
```

On Release 6, it is passed an `AuthHeader` structure:

```
Pdu *parse_pdu(AuthHeader *auth_ptr);
typedef struct _AuthHeader {
     OctetString *packlet;
     unsigned long version;
     OctetString *community;
} AuthHeader;
```

The routines are identical for applications compiled on Release 5 with SNMPV1 or SNMPV1_ONLY set.

2.2.13 SNMP I/O (libsnmpio) interface

The version of *libsnmpio* supported on Release 6 implements SNMPv1.

The Release 5 version of *libsnmpio* (included in */osr5/usr/lib*) is supported on Release 6 for Release 5 binaries only.

The header files for SNMP are found in */usr/include/snmp* on Release 5, and in */usr/include/netmgt* on Release 6.

2.2.13.1 get_response

This routine is defined on Release 5 as follows:

```
int
get_response(fd, src, in_packet, in_packet_len, timeout);
int fd;
struct sockaddr_in *src;
u_char *in_packet;
long *in_packet_len;
int timeout;
```

It is defined on Release 6 as:

```
int get_response(int seconds);
```

8.2.11.2 initialize_io

This routine is defined on Release 5 as follows:

```
int
initialize_io(program_name, name, sin);
char *program_name;
char *name;
struct sockaddr_in *sin;
```

It is defined on Release 6 as:

```
void initialize_io(char program_name, char name);
```

2.2.13.2 send_request

This routine is defined on Release 5 as follows:

```
int
send_request(fd, dst, out_packet, out_packet_len);
int fd;
struct sockaddr_in *dst;
u_char *out_packet;
long out_packet_len;
```

It is defined on Release 6 as:

```
int send_request(int socket, AuthHeader auth_pointer);
```

2.2.14 SMUX (libsmux) interface

The version of *libsmux* supported on Release 6 implements SNMPv1.

The Release 5 version of *libsmux* (included in /osr5/usr/lib) is supported on Release 6 for Release 5 binaries only.

The header files for SNMP are found in */usr/include/snmp* on Release 5, and in */usr/include/netmgt* on Release 6.

Source compatibility for every one of these calls is affected by differences in the data structures used by the Release 5 and Release 6 implementations. See the following sections for details:

• ‘‘Object identifier (OID) structure’’

• ‘‘Object type (OT) structure’’

• ‘‘SCO OpenServer 64 bit counters’’

• ‘‘Aggregate structure’’

2.2.14.1 Object identifier (OID) structure

On Release 5, the `length` element of the `OID` structure is a `long`, on Release 6 it is a `short`.

2.2.14.2 Object type (OT) structure

The `object_type` (OT) structure is defined differently on Release 5 and Release 6.

The OT structure on Release 5 is defined as follows:

```
typedef struct object_type {
     char *ot_text; /* OBJECT DESCRIPTOR */
     char *ot_id; /* OBJECT IDENTIFIER */
     OID ot_name; /* .. */
```

```
    OS ot_syntax; /* SYNTAX */
    int ot_access; /* ACCESS */
    #define OT_NONE 0x00
    #define OT_RDONLY 0x01
    #define OT_WRONLY 0x02
    #define OT_RDWRITE (OT_RDONLY | OT_WRONLY)
    #define OT_RDCREAT (0x04 | OT_RDWRITE)
    u_long ot_views; /* for views */
    int ot_status; /* STATUS */
    #define OT_NONE 0x00
    #define OT_OBSOLETE 0x01
    #define OT_CURRENT 0x02
    #define OT_OPTIONAL 0x03
    #define OT_DEPRECATED 0x04
    caddr_t ot_info; /* object information */
    ot_getfunc ot_getfnx; /* get/get-next method */
    ot_setfunc ot_setfnx; /* set method */
    #define type_SNMP_PDUs_commit (-1)
    #define type_SNMP_PDUs_rollback (-2)
    caddr_t ot_save; /* for set method */
    int ot_range; /* close enough */
    int ot_lendpoint; /* .. */
    int ot_rendpoint; /* .. */
    char *ot_index; /* INDEX */
    char *ot_augments; /* or AUGMENTS */
    caddr_t ot_iit; /* .. */
    caddr_t ot_smux; /* for SMUX */
    struct object_type *ot_chain; /* hash-bucket for text2obj */
    struct object_type *ot_sibling; /* linked-list for name2obj */
    struct object_type *ot_children; /* .. */
    struct object_type *ot_next; /* linked-list for get-next */
} object_type, *OT;
```

On Release 6, the structure is declared as follows:

```
typedef struct object_type {
    char *ot_text; /* OBJECT DESCRIPTOR */
    char *ot_id; /* OBJECT IDENTIFIER */
    OID ot_name; /* .. */
    OS ot_syntax; /* SYNTAX */
    int ot_access; /* ACCESS */
    #define OT_NONE 0x00
    #define OT_RDONLY 0x01
    #define OT_WRONLY 0x02
    #define OT_RDWRITE (OT_RDONLY | OT_WRONLY)
    unsigned long ot_views; /* for views */
    int ot_status; /* STATUS */
    #define OT_OBSOLETE 0x00
    #define OT_MANDATORY 0x01
    #define OT_OPTIONAL 0x02
    #define OT_DEPRECATED 0x03
    caddr_t ot_info; /* object information */
```

```
        IFP ot_getfnx; /* get/get-next method */
        IFP ot_setfnx; /* set method */
        #define type_SNMP_PDUs_commit (-1)
        #define type_SNMP_PDUs_rollback (-2)
        caddr_t ot_save; /* for set method */
        caddr_t ot_smux; /* for SMUX */
        struct object_type *ot_chain; /* hash-bucket for text2obj */
        struct object_type *ot_sibling;/* linked-list for name2obj */
        struct object_type *ot_children; /* .. */
        struct object_type *ot_next; /* linked-list for get-next */
    } object_type, *OT;
```

In addition, the OS structure has some extra elements on Release 5:

```
typedef struct syntax {
    char *os_name; /* syntax name */
    int os_type; /* syntax type */
    os_decode_func os_decode; /* PE -> data */
    os_free_func os_free; /* free data */
    char *os_textual; /* for textual conventions */
    char *os_display; /* for textual conventions */
    struct enum_syntax *os_enum; /* for enumerations */
} *OS;
```

The Release 6 version of the OS structure is:

```
typedef struct object_syntax {
char *os_name; /* syntax name */
int os_type; /* syntax type */
IFP os_decode; /* PE -> data */
IFP os_free; /* free data */
} *OS;
```

2.2.14.3 SCO OpenServer 64 bit counters

Objects of type `Counter64` and the `Counter64` structure are not supported on Release 6, but are supported on Release 5.

The Release 5 `ObjectSyntax` structure has an extra `c64_value` element not supported on Release 6.

2.2.14.4 Aggregate structure notes

A number of aggregate structures, such as those listed below, are affected by changes to lowerlevel structures.

For example, the OI structure is declared identically in both Release 5 and Release 6; but the structure members are of type OID and OT, which do have implementation differences.

The affected data structures are:

- `OI`/`OIDentifier`

- all types of the form `type_SNMP_VarBind`, `type_SNMP_VarBindList`, `type_SNMP_ObjectName`, `free_SNMP_ObjectName`, `type_SNMP_ObjectSyntax`, and `free_SNMP_ObjectSyntax` and any structures declared of these types
- `VarBindList` and `VarBindUnit`
- `Pdu`, and all types ending in `PDU` and `PDUs`

See the header file */usr/include/netmgt/snmp.h* and the sections ‘‘Object identifier (OID) structure’’, ‘‘Object type (OT) structure’’, and ‘‘SCO OpenServer 64 bit counters’’.

2.2.15 Termios and termio interfaces

The **termios** interface is the preferred API for terminal management functions; the **termios** calls are translated into **ioctl** calls that issue the appropriate commands for the given operation.

Applications should never directly issue **ioctl** commands to terminal devices, but should use the **termios**(S) functions instead. Binaries produced using the Release 5 compilers will get iBCS2-compatible behavior when run on Release 6.

Source code from Release 5 will need to be changed to use the `termios` structure supported on Release 6, which omits the `c_line` (line discipline) element supported on Release 5, and uses a different value for the number of elements in the control character (`c_cc`) array.

2.2.16 curses (libocurses) interface

On Release 6 and Release 5, the standard **curses** library is the UNIX System V Release 4 (SVR4) curses library, *libcurses.a*. OSR5 ABI binary applications that use the standard **curses** libraries on those systems will execute as expected on Release 6.

While all the same function names are supported and these functions all have the same semantics on the two systems, there are slight differences in the:

- header files
- **terminfo** and **termcap** databases

These are detailed in the next few sections.

2.2.16.1 curses header files

On Release 6, the SVR4 curses interfaces are defined in */usr/lib/ocurses.h*.

On Release 5, *curses.h* is a wrapper that reads in either of *tcap.h* or *tinfo.h*, depending on whether the application is compiled with **termcap** or **terminfo** support (**termcap** is not supported on Release 6). Since most applications are simply compiled with `#include curses.h`, this should not present a compatibiliy problem unless the lower level header files are included directly.

2.2.16.2 terminfo and termcap databases

These files define terminal characteristics and are used by various libraries and programs (for example, the **curses** library routines and the **vi** editor) to perform screen management functions.

The location and format of the **terminfo** and **termcap** databases are the same on all systems.

There are no known compatibility impacts for these databases.

2.2.16.3 X/Open curses (formerly libstdcurses) library

This library (*libocurses*) is supported on Release 6, but not on Release 5.

2.2.17 BSD database management (libdbm and libndbm) interface

Release 5 provides database management routines in the *libdbm* and *libndbm* libraries. Release 5 binaries that use these libraries will load and run correctly on Release 6.

For more information on using BSD compatibility libraries on Release 6, see ‘‘BSD system libraries and header files’’.

2.2.18 Encryption (libcrypt) interface

The *libcrypt* interface is documented on **crypt**(S). The **crypt**, **encrypt**, and **setkey** routines are also declared in *libgen* and in *libc*.

2.2.19 Executable and Linking Format (libelf) interface

The *libelf* implementations on Release 5 and Release 6 are nearly identical, with the following exceptions:

- Release 6 provides an additional flag, ELF_C_IMPURE_WRITE, that can be passed to **elf_begin**(ELF). Portable applications and applications targeted for Release 5 should not use this flag, as it is not supported on Release 5.
- The **nlist** function is declared in *libelf* on Release 6 and *libc* on Release 5. For Release 5 applications run on Release 6, the Release 6 kernel uses the Release 5 *libc*.

The Release 6 *libelf* interface is documented on the Section ELF manual pages.

2.2.20 Graphic interfaces

- Graphical programs sometimes rely on Motif 2.x. This version of Motif is normally compatible with Motif 1.2 (included with SCO OpenServer 6). In case problems do occur, you can try using Lesstif (available from **http://www.lesstif.org**).
- A lot of graphical GNU programs use the **xpm** package, support for which is not provided by SCO OpenServer 6. The best thing to do is download the **xpm** source and build it.

3 A Guide to debug for dbx Users

This section details feature differences between dbx and debug, and offer help for those users who are used to a dbx-style debugger. For simplicity of presentation, we concentrate on the command line and do not discuss the differences between the graphical front-ends for the debuggers. Extra features offered by debug are detailed in the **debug**(CP) manual page and the **debug help** command.

3.1 Starting debug

Two common methods of starting **debug**:

1. From the command line you can specify:

   ```
   debug arglist
   ```

   The arglist must contain an absolute or relative pathname to a binary file and the appropriate arguments for that binary. Then, the debug internal command

   ```
   run [-f]
   ```

   starts execution of the binary.

2. If you start debug with no arglist, then enter these commands at the debug prompt to start execution:

   ```
   debug> create pathname_to_binary [arg...]
   debug> run [-f]
   ```

The important thing to remember when specifying the pathname to the binary is that the PATH variable is not used to search for it; so, you must use a full pathname for any binary not in the current working directory (or below).

3.2 Command Line Options

dbx option	debug option	Description	Notes
corefile	-c corefile	File name of the core file to be debugged	
-r	N/A	Executes the object file and exits the debugger only if no error occurs	
-x	N/A	Ignore cross-reference file when initializing dbx	Not currently available in debug.

dbx option	debug option	Description	Notes
-F	N/A	dbx assumes that file scoped structure, union, enum definitions with the same name are identical	Unnecessary
-I dir	-s path	Search path(s) for source files	Under debug this sets the global path variable, %global_path
-c file	N/A	Execute the dbx commands in the file before reading from standard input	debug offers the script internal command to do this from the debug prompt
-C++	N/A	Assume C++ mode	Set %lang to “C++” at the debug prompt to override the default value in %db_lang.
-C	N/A	Use C debugging only	Set %lang to “C” at the debug prompt to override the default value in %db_lang.

3.3 Setup

dbx	debug	Description
$HOME/.dbxtrarc	$HOME/.debugrc	File containing commands that are executed at debugger startup

3.4 Configuration example

Here is a useful set of entries to put in $HOME/.debugrc to tailor debug commands and initial behaviour to be similar to dbx.

```
#---- UW7 debug ~/.debugrc file for migrating from dbx style usage ----
#
# default cmdline editing
set %mode “emacs”
# where/stack trace
alias w stack
# Continue
alias c run -f
# swap +/- up/down for gdebug - depends on %frame_numbers
alias up set %frame %frame +1; list -c 1
alias down set %frame %frame -1 ; list -c 1
# status
alias st events
#remove existing alias
```

```
alias unalias alias -r
# breakpoints
#alias b stop
alias d delete
alias dis disable
alias di disable
alias en enable
# rerun - load and run
alias load create
unalias rr
alias rr create -f none $* ; run -u main
# general useful
alias a alias
unalias h; alias h history
alias H help $* |$PAGER
alias P $* |$PAGER
alias mem dump -b
alias echo {print -f "%s\n" $*; }
#disassemble with as much info as poss
set %dis_mode="source"
alias dasm dis
# don't follow any forked processes - does this actually work ??
set %follow "none"
set %thread_change="ignore"
#set %thread_change="announce"
# shutup signals we don't normally care about
signal -d -i sigcld sigalrm
# Where to look for src
set %global_path="/tmp/vtcl"
alias path {if ($# > 0) set  %global_path $* ""; print %global_path; }
#
# User variables
set $MALLOC_CHECKS=6
set $MALLOC_STATS=6
# into env
alias EMC export $MALLOC_CHECKS ; stop malloc.c@checkmsg
alias EMS export $MALLOC_STATS
```

3.5 General Tips

The following are general usage differences between dbx and debug:

- For help on any command or usage of debug, use the help command to display a list of commands and internal help topics (such as for help on specific debug internal variables). Some examples:

```
help
help expr
help release
help %db_lang
help C++
```

- debug internal variables are preceded with a **%** character. So **%lang** is used to identify the **debug** internal variable for setting the current language.
- **debug** command line editing can be improved by switching it to **vi** or **emacs** mode by setting the **%mode** variable (for example: **set %mode=vi**).
- The debug command line can have redirection symbols and pipes used with any command. dbx only offers redirection with certain commands such as run and dump.
- You can specify breakpoints on static fns (or reference static data) but you must explicitly specify the filename the fn/variable resides in in the expression or already have the debugger stopped in the file in question.

```
stop malloc.c@checkmsg
stop file.c@static_fn
print file.c@static_data
```

- All signals are trapped by default (suppress **sigcld**, **sigalrm**):

```
signal -d -i sigcld sigalrm
```

  and forked processes are followed by default.

```
{debug|create} -f none
run -f
set %follow “none”
release
```

- When a process is not loaded or a process is completed, many commands that are irrelevant to a process are disabled along with those that are not. For example: listing a file, and listing breakpoints.
- debug runs by default in its own X window; suppressable with -ic or by unsetting $DISPLAY; not specifiable from defaults file so can tweak it permanently. You can use a shell alias; for example:

```
alias dbg='debug -ic'
```

- In the command **debug *cmdline***, the first token is a filename that is not searched for in PATH; i.e., it must be specified as an absolute (or relative) pathname if the binary is not in the current working directory. This also applies to the internal **create** command.
- **Print “v”** -> outputs “v” (quoted - instead of v)
- **^D** doesn’t exit the debugger; use **quit** or **exit**.
- debug uses two expression parsers: one for the command line constructs and another for C/C++ expressions. The combination of the two can lead to unexpected effects and demand non-obvious syntax (e.g., empty string inclusion).
- Aliases are set and removed in unique ways (that is, more like macro replacement than other assignments). To set:

```
alias name list_of_replacement_tokens
```

To remove:

```
alias -r
```

- Substituting args are not expanded inside quoted strings or specifiable with brackets or parentheses (in disambiguating ${} forms):

```
$[0-9]
$*
$#
```

- In debug, use aliases and blocks (enclosed in brackets: {...}) to define user functions.
- By default debug will attempt to also debug forked child processes. The %follow system variable controls this behaviour and should be set to none (from the default all) to disable this.
- The threads of control of a process that debug knows about can be displayed with the ps command and the current switched by setting the %proc system variable or the -p switch on many commands.

  Notification of thread state change (create, exit, suspend, continue, etc.) can be controlled with the %thread_change variable. The default value is stop, indicating to print a message and stop the thread if possible. To disable both notification and stopping set it to ignore.

3.6 Debugger Variables

The **debug**-specific variables are substantially different from those with special meaning for dbx. There are four types of debug variables, each with a distinct purpose:

- ‘‘code’’ variables; variables defined by the code being debugged (unadorned name): **var**
- ‘‘system’’ variables; variables that affect or modify some aspect of debug behaviour: **%var**
- ‘‘user’’ variables; generic debug internal variables defined by the user to store temporary values: **$var**
- environment variables are handled as a subset of user variables, and are defined by exporting them: **export $var**

3.7 Execution and Tracing Commands

dbx command	**debug** command	Description	Notes
run [args] [< filename][>[> filename]]	**run** [-p proc_list][-bfr][-u location]	Run the current program being debugged	Debug treats the run command also as continue, so by using run when the execution of a program has been halted will continue from that point. This is different from dbx's run as it restarts program. Note that the redirection symbols in debug can be used with any command. Debug also offers pipes.
rerun [args] [< filename][>[> filename]]	**create** [-f none | procs | all] [-dr][-l start_loc] [command_line]	Rerun the program and append the arguments to the previous arguments passed to the program Debug's create command (with no arguments) offers a similar functionality to dbx's rerun command. Note that debug does not append given arguments with the create command, but treats it as a request to debug a new program.	
setenv name value	**set** [-p proc_list] debug_or_user_var [=] expr [,expr] export $username	Set the value of an environment variable in the environment of the debugged program.	In debug, the set command must be used with a user variable to set the value and then exported to the environment of the debugged program with the export command. Note that this has to be done before the execution of the debugged program.
unsetenv name	N/A	Unset the value of the variable.	
trace [inst][in routine][if condition] **trace** [inst] line-number|address [if condition] **trace** routine [in routine2][if condition]	(various)	Display source lines (or machine instructions) when executed, display the arguments and results of routines	There are a number of ways of emulating the trace functionality. Aliases can be created on the variations of: **alias trace = onstop { step }** **debug variables %func, %frame**, etc. can be used to determine the current execution location.

dbx command	**debug** command	Description	Notes
trace [inst][change] variable [in routine][if condition] **trace** [inst] change address [in routine][if condition] **trace** [inst] access variable [in routine][if condition] **trace** [inst] access address [in routine][if condition]	(various)	Display when changes (or accesses) are made to variables or memory locations.	This functionality can be reasonably matched by debug's **stop** command used with expressions.
stop in routine [if condition] **stop** [inst] access variable [if condition] **stop** [inst] [change] address [if condition] **stop** [inst] [change] variable [if condition] **stop** [inst] if condition **stop** [inst] at line_number|address [if condition] **stop** [inst] access address [if condition]	**stop** [-p proc_list] [[-q] [-c count] stop_expr]	Stop execution when execution reaches a given location, or when a variable or memory location is changed or accessed.	All the various forms of the dbx stop command can be matched by the debug stop command used with expressions.
when [inst] at line_number|address [if condition] {command; [command; ...]} **when** in routine [if condition] {command; [command; ...]} **when** [inst] condition {command; [command; ...]} **when** [inst] change variable [if condition] {command; [command; ...]} **when** [inst] change address [if condition] {command; [command; ...]} **when** [inst] access variable [if condition] {command; [command; ...]} **when** [inst] access address [if condition] {command; [command; ...]}	**stop** [-p proc_list] [[-q] [-c count] stop_expr [command]]	Execute a series of commands when execution reaches a given location, when a condition is true, or when a variable or memory location is changed or accessed.	

dbx command	**debug** command	Description	Notes
status [> filename]	**events** [-p proc_list] [event_num ...]	List the user specified events for the current program (break points, etc..) and their current status.	The debug events command also provides information about system calls, signals and exceptions)
delete event_num ...	**delete** event_num ...	The event corresponding to event_num is removed.	Debug provides another variation of this command for deletion of event types.
disable event_num ...	**disable** event_num ...	Disable the specified events.	Debug provides another variation of this command to disable event types.
enable event_num ...	**enable** event_num ...	Enable previously disabled events.	Debug provides another variation of this command to enable event types.
catch number|signal_name	**signal** -d [signal ....] **signal** [-p proc_list][[-q] signal ... [cmd]]	Trap a specified signal. Debug offers greater control over signals. Refer to the relevant documentation.	
ignore number|signal_name	**signal** -d -i [signal ....] **signal** [-p proc_list] [[-iq] signal ... [cmd]]	Stop trapping a specified signal.	Debug offers greater control over signals. Refer to the relevant documentation. Note that the -i option can be also used to re-establish the default action for a signal.
cont integer	**run** [-p proc_list][-bfr][-u location]	Continue execution from where the program stopped.	
cont signal_name	**kill** [-p proc_list][signal] followed by **run** [-p proc_list][-bfr][-u location]	Process continues as though it received the given signal	Debug needs to have the run command executed after the kill command is given.
skip [n]	(alias)	Continue execution from where stopped. If n is specified, that many breakpoints are ignored before the program stops. If n is not given, one breakpoint is skipped.	This functionality can be emulated with debug's onstop command, e.g. if ($# == 0) set $skip=1; else set $skip=$1 +0; onstop { if ($skip > 0) { set $skip=$skip - 1; run } else delete %thisevent}; run Note that the "+0" is required when no parameters are passed to skip.
step [n]	**step** [-p proc_list] [-bfq] [-c count]	Execute a number of source lines (default is 1)	Debug provides more control to the user for stepping.

dbx command	**debug** command	Description	Notes
next [n]	**step** [-p proc_list] [-bfq] -o [-c count]	Execute a number of source lines (default is 1). If a line contains a call to a procedure or function the command does not stop at the beginning of that block. Debug provides more control to the user for stepping. Note that there is a standard alias, next, for debug.	
return [procedure]	**run** -r	Continue until a return to procedure is executed, or until the current procedure returns if none is specified. Debug's run -r continues execution until a the return address of the current function is reached. Note that you cannot specify a procedure name with the -r option.	
call procedure(parameters)	(various)	Execute the object code associated with the named procedure or function.	A call to a function or procedure under debug can be done using any command that allows an expression for an argument. For example: print procedure(a, b, c) or set $MYRET=function(a, b).
jump line_number	**jump** [-p proc_list] location	Continue until a return to procedure is executed, or until the current procedure returns if none is specified.	Debug's jump command can be used with any valid location.
goto line_number	**run** -u location	Continue execution and stop before the fist instruction at the specified line number is executed.	Debug's run command with the -u option can be used with any valid location.

3.8 Printing Variables and Expressions

Please read **debug**(CP) or use the **debug**

```
help expr
```

internal command for specific information about debug expressions.

dbx command	debug command	Description	Notes
assign variable=expression	**set** [-p proc_list] [-v] debug_or_user_var [=] expr [,expr...] **set** [-p proc_list] [-v] language_expression	Assign the value of an expression to a variable.	Debug's set can be used to evaluate any language expression, typically and assignment.
assign register=expression assign eax=6	**print** %eax=(void *)6	Assign the value of an expression to a register.	The type casting is required; otherwise debug returns a warning which is not otherwise suppressible.
assign memloc=expression **assign** 0x8049478=5	**print** *(int *)0x8049478=5	Assign the value of an expression to a memory location.	The type casting is required; otherwise debug returns a warning which is not otherwise suppressible.
dump [procedure] [> filename]	**functions** [-s] [-p proc_list] [-o object] [-f filename] [pattern] **symbols** [-p proc_list] [-o object] [-n filename] [-dfgltuv] [pattern] Print the names and values of variables in a procedure.	Debug's functions and symbols commands can provide information to match dbx's dump command and more.	
print expression [, expression ...]	**print** [-p proc_list] [-f format] [-v] expr, ...	Print the expression.	The -f format option to debug's print command allows the user to use a "C" style format output of the expression, e.g. print -f "0x%x\n" main
whatis expression	**whatis** [-p proc_list] expr	Print the declaration of the expression.	Debug's whatis command provides all known information about the expression.
which identifier	N/A	Print the full qualification of the given identifier.	Unavailable cross reference information required for this command.

dbx command	debug command	Description	Notes
up [count] **down** [count]	**set** %frame = frame_number	Move the current frame up and down the stack.	Debug's %frame variable contains the current frame number. By setting this variable a user can traverse the stack. These can be aliased with: alias up set %frame = %frame + 1 alias down set %frame = %frame - 1
where identifier	**stack** [-p proc_list] [-f frame] [-c count] [-a address] [-s stack]	Print the list of the active procedures and functions. Debug's stack command offers more functionality that dbx's where command.	
whereis identifier	N/A	Print the full qualification of all the symbols whose name matches the given identifier.	Unavailable cross reference information required for this command.
dbxref [-i][-o file][options][files]	N/A	Generate the cross-reference file for dbx's xref, whatis and whereis commands.	
xref identifier	N/A	Print a cross-reference for the given symbol. Unavailable cross reference information required for this command.	

3.9 Accessing source files

dbx command	**debug** command	Description	Notes
/regular expression[/] ?regular expression[?]	**list** [-p proc_list] [-c count] /regexp/ **list** [-p proc_list] [-c count] ?regexp?	Search forward or backward in the current source file for the given pattern.	In debug / and ? when %mode is set to vi will search forward and backward in the command line history.
edit [filename] **edit** procedure_name **edit** function_name	N/A	Invoke an editor on filename or the current source file if none is specified.	The shell command (!) with the appropriate executable can be used to edit a file.
file [filename]	**set** %list_file=filename	Change the current source file name to filename.	Debug's current file is also contained in %file
func [procedure/function]	N/A	Change the current function.	
list . **list** procedure/function **list** [source_line number [, source_line_number]]	**list** [-p proc_list] [-c count] [thread id@][file_name@][header_file@]func_name **list** [-p proc_list] [-c count] [thread id@][file_name@][header_file@]line_num **list** [-p proc_list] [-c count]	List the lines in the current source file.	
use directory_list	**set** %path="directory_list" **set** %global_path="directory_list"	Set the list of directories to be searched when looking for source files.	

3.10 Command Aliases

dbx command	**debug** command	Description	Notes
alias **alias** name name **alias** name "string" **alias** name (parameters) "string"	**alias** name tokens ... **alias** [name]	Establish a new alias.	Debug alias parameters are available as $1, $2, etc... $# is the number of arguments passed and $* represents all the arguments passed.
unalias name	**alias** -r name	Remove the alias with the given name.	

dbx command	**debug** command	Description	Notes
set name [= expression]	**set** [-p proc_list] [-v] debug_or_user_var [=] expr [,expr...	Define the values for debugger variables.	Debug specific variables are preceded with a %, e.g. %file. User specific (and environment) variables are preceded with a $, e.g. $DISPLAY.
unset name	N/A	Delete the debugger variable associated with name.	

3.11 Machine Level Commands

dbx command	**debug** command	Description	Notes
stepi	**step** [-p proc_list] [-bfq] -i [-c count]	Single step as in step, but at a single machine instruction.	Debug has a default alias for this functionality, si.
nexti	**step** [-p proc_list] [-bfq] -io [-c count]	Single step as in next, but at a single machine instruction.	Debug has a default alias for this functionality. ni.
regs	**regs** [-p proc_list]	Display the contents of the register set.	
fpregs	**regs** [-p proc_list]	Display the contents of the register set. Debug display's the floating point registers only if they have been used.	
address, address/ [mode] **address** / [count] [mode]	**dump** [-p proc_list] [-c byte_count] [-b] expression	Print the contents of memory	Debug's dump command allows starting points to be defined by expressions as well as addresses. Debug's dump output is printed in ASCII and hexadecimal.

3.12 Miscellaneous Commands

dbx command	**debug** command	Description	Notes
cc	**set** %db_lang="C" or **set** %db_lang="C++"	Toggle between C++ and C output modes.	
help [command] help [string]	**help** [topic]	Obtain a list of help topics or help on a topic.	
quit	**quit**	Exit the debugger.	Debug's quit kills all created processes or releases them if they had been grabbed.
sh [command_line] ! command_line	**!** command_line !!	Pass the command line to the shell for execution.	Debug's !! command executes the last shell command.
source filename	**script** [-q] filename	Read debugger commands from a file.	
kill	**kill** [-p proc_list]	Kill the current or specified process. Debug's kill command can also be used to send signals to a process.	

dbx command	**debug** command	Description	Notes
detach	**release** [-s] [-p proc_list]	Release the current process. Debug's release command allows the process to be released in a stopped state with the -s option.	
exec filename	(automatic)	If the process being debugged does an exec system call, this tells the debugger that a new symbol table should be read in.	Debug is capable of debugging several processes and threads at the same time, so this functionality is executed automatically.
version	**version**	Print the version information for the debugger.	

3.13 Common tasks

Task	**dbx** cmd	**debug** equiv
Breakpoints		
setting	**b, break, bp**	**stop**
clearing	**d, del**	**delete**
showing	**status, st**	**events**
temp disable	**dis**	**disable**
reenable	**en**	**enable**
Single stepping		
into fns	**s, step**	**step, s**
over fns	**n, next**	**step -o, n, next**
Running	**run, r**	**run [-f]**
rerunning process	**rerun, rr**	**create [-f none]; run none**
Continuing	**c, cont**	**run [-f]**
Viewing		
print variables	**print**	**print, p**
dump memory	**mem, print**	**dump locn**
stack trace	**where**	**stack [-c count]**

These are perhaps less commonly used but usually still available in some form:

Task	**debug** equiv
Current Stack frame	**modify** %frame system variable: **alias** down set %frame = %frame - 1; list -c 1 **alias** up set %frame = %frame + 1; list -c 1
Setting variables	set v[=]value
Source	
Specifying where the source files are	**%global_path** - all processes **%path** - for a single process **set** -p <proc> %path whatever - after process is running

Task	**debug** equiv
Changing srcFile to Browse	**set** %list_file filename
Execution	
run to return fm current fn	**run** -r
execute from an address	**jump** loc
run upto address	**run** -u loc
rerun current process	**create** [-f none]; run [-f]
viewing/changing *argv* list	**rerun** and specify a new command line
enabling command line editing	**set** %mode vi **set** %mode emacs (emacs mode deficient - no handling arrow keys; occasional dropout)
Displaying paged output	(**help**) must pipe through pager: ... | **more**
Watchpoints	not provided, but buildable with onstop command