

ObjectCenter Platform Guide for SUN and HP Users

ObjectCenter Version 2.2.0

Table of Contents

- **SUN System Requirements**
 - Supported SUN Platforms
 - Supported SUN Compilers
 - Supported Windowing Systems

- **HP System Requirements**
 - Supported HP Platforms
 - Supported HP Compilers
 - Supported Windowing Systems

- **SUN C Library Functions Replaced**

- **HP C Library Functions Replaced**

- **SUN Shared Libraries**
 - Search Rules for Loading Libraries
 - Using Environment Variables to Modify Loading Libraries
 - Using Switches to Modify the Loading of Libraries
 - Specifying the Binding Mode
 - Loading the Static Version of the C and C++ Libraries
 - Setting Breakpoints in Shared Libraries while in PDM
 - Binding of Functions and Data
 - Unloading a Specific Function
 - Other SunOS (Solaris 1) Differences
 - Unloading Shared Objects
 - Unloading a Specific Module
 - Using Initialized Global Data

- **Using HP Shared Libraries**

- [Loading Shared Libraries](#)
 - [Search Rules for Loading Libraries](#)
 - [Loading the Archive Version of a Library](#)
 - [Specifying Global ld Switches with LDOPTS](#)
 - [Binding of Functions and Data](#)
 - [Unloading a Specific Function](#)
 - [Shared Library Stack Frames in Core Files](#)
 - [Link with libCdynamic to Invoke Static Initializers](#)
 - [Calls to shl_* \(3X\) Functions](#)
- **[Potential SUN Anomalies](#)**
 - [Default Parser Configuration](#)
 - [Sun C Compilers and pdm](#)
 - [Switches and Variables Ignored by load \(SunOS/Solaris 1\)](#)
 - [Instrumented Object Code](#)
 - [Setting LD_LIBRARY_PATH](#)
 - [Locating X11 Header Files](#)
 - [Resource Limits for Stacksize](#)
- **[Potential HP Anomalies](#)**
 - [Default Parser Configuration](#)
 - [Supported cc and c89 Switches](#)
 - [Supported ld Switches](#)
 - [Different Behavior of the load Command](#)
 - [Unsupported Environment Variables](#)
 - [No PIC Support](#)
 - [No Support for Cache Hint Bits](#)
 - [void Types may be Misinterpreted](#)
 - [No Support for Long Pointers](#)
 - [No Support for Argument Values in Intrinsic](#)
 - [ObjectCenter Replaces Fewer Header Files](#)
 - [Changes in Signal Handling](#)
 - [sigvec\(\) and sigvector\(\) Flags](#)
 - [sigaction\(\) Flags Ignored](#)
 - [No sigspace\(\) and No sigstack\(\)](#)
 - [_longjmp and _setjmp](#)
 - [Enhanced sigset\(\) and sigvector\(\)](#)
 - [syscall\(\) Function](#)
 - [ioctl\(\) Support](#)
 - [Static Symbol not in symtab](#)
 - [Attaching to a Running Process may Fail on HP-UX](#)
 - [clcc Command-Line Switches](#)

This Platform Guide describes the supported compilers, supported windowing systems, library functions replaced by ObjectCenter for run-time error checking, shared library support, and potential SUN and HP anomalies when using ObjectCenter version 2.2.0.

For ObjectCenter's memory, swap space, and disk requirements, please see the [ObjectCenter Version 2.2.0 Release Bulletin](#) on the main [ObjectCenter page](#) of CenterLine's website.

SUN System Requirements

This version of ObjectCenter supports the following platforms, compilers and windowing systems for SUN architectures.

Supported SUN Platforms

For a list of the platforms supported by ObjectCenter Version 2.2.0, refer to the "Supported Platforms" section of the [ObjectCenter Version 2.2.0 Release Bulletin](#). Additionally, you can review CenterLine's [Product Compatibility Matrix](#) which summarizes platform, compiler, etc. support for all CenterLine products.

NOTE: Please refer to your [Release Bulletin](#) for information about Sun patches that should be installed before you use ObjectCenter.

NOTE: Before installing ObjectCenter on SunOS 4.x systems, make sure that the operating system was installed with the System V software installation option. This is necessary because the ObjectCenter C++ compilation system depends on the **cut** and **expr** commands that are supplied only with the System V installation option.

Supported SUN Compilers

ObjectCenter supports the following compilers on SunOS 4.x and Solaris 2.x operating systems:

- CenterLine-C compiler (clcc)
- Sun K&R C compiler (cc), all versions
- Sun ANSI C compiler (acc), all versions
- SPARCompiler C (ANSI C) (acc or cc), all versions
- ObjectCenter's C++ compiler (CC)

ObjectCenter supports the following compilers on SunOS and Solaris but with limitations to browsing and source level debugging. CenterLine-C and ObjectCenter's C++ compiler (CC) are link compatible with them.

- GNU C compiler (gcc), all versions **UP THROUGH AND INCLUDING version 2.5.8 ONLY.**
- FORTRAN
- Sun C++ compiler (**cfront-based**), Version 3.0.1 (CC) **ONLY**

The Sun C++ Compiler Version 3.0.1 produces ANSI C intermediate code. You may wish to set the ObjectCenter **backend_ansi** option to cause ObjectCenter to generate ANSI C intermediate code for improved capability. Please refer to the **code generation** entry in the *ObjectCenter Reference* for information about the ObjectCenter **backend_ansi** switch and option.

This version of ObjectCenter supports Release 3.0.2.15 of the USL C++ Language System, which is backward compatible with Release 2.x and 3.0.x. We support everything in the USL C++ release, except that we do not support or supply the **task** library (**libtask.a**).

In most cases, code that compiled without warning in USL C++ Release 2.x or 3.0.x will compile correctly in Release 3.0.2.15. However, you may run across some cases that cause problems. For instance, you may find that some cases of **switch** statements fail to compile, complaining about jumping past an initializer; the fix here is to supply curly braces ({}) around the body of the **case** statement. For best results, put the **break** statement, if any, outside the closing curly brace.

You may also encounter an "**unresolved symbol**" problem at link time if you have link symbols that contain nested types, including nested **enums**. USL C++ Release 2.x, 3.0.x and 3.0.2.15 encode the names of such symbols differently. The fix is to recompile the problem modules with **cfront** 3.0.2.15. Fortunately, this situation is uncommon.

Supported Windowing Systems

ObjectCenter supports both the Motif and OPEN LOOK windowing systems on SunOS and Solaris platforms. OPEN LOOK is the default on the Sun platform. You can choose Motif at startup with the **-motif** switch on the **objectcenter** command line.

ObjectCenter also supports the Common Desktop Environment (found on some SUN platforms).

HP System Requirements

This version of ObjectCenter supports the following platforms, compilers and windowing systems for HP architectures.

Supported HP Platforms

For a list of the platforms supported by ObjectCenter Version 2.2.0, refer to the "Supported Platforms" section of the **ObjectCenter Version 2.2.0 Release Bulletin**. Additionally, you can review CenterLine's **Product Compatibility Matrix** which summarizes platform, compiler, etc. support for all CenterLine products.

Supported HP Compilers

ObjectCenter supports the following compilers on HP-UX operating systems:

- CenterLine-C compiler (clcc)
- HP C compilers (cc and c89), all versions
- ObjectCenter's C++ compiler (CC)
- HP C++ compilers (**cfront-based** versions **ONLY**)
- Softbench C compiler

ObjectCenter supports the non-ANSI, cfront-based HP C++ compiler but

with the following limitations:

- The HP C++ compiler **+eh** switch (for exception-handling) must not be used.
- HP C++ debugging information is not processed, so source level debugging and browsing of HP C++ object code is restricted.

Code generated by CenterLine C and C++ compilers is link-compatible with code generated by HP compilers, as long as the HP object code was not compiled with the **+eh** switch.

You can also load **FORTRAN** object files that are in either of the following categories:

- Object code compiled without debugging information
- Object code compiled with debugging information but loaded with the **-G** switch.

ObjectCenter on the HP platform does **NOT** support source-level debugging of object modules compiled by **gcc**, the GNU C compiler available from the Free Software Foundation.

This version of ObjectCenter supports Release 3.0.2.15 of the USL C++ Language System, which is backward compatible with Release 2.x and 3.0.x. We support everything in the USL C++ release, except that we do not support or supply the **task** library (**libtask.a**).

In most cases, code that compiled without warning in USL C++ Release 2.x or 3.0.x will compile correctly in Release 3.0.2.15. However, you may run across some cases that cause problems. For instance, you may find that some cases of **switch** statements fail to compile, complaining about jumping past an initializer; the fix here is to supply curly braces ({}) around the body of the **case** statement. For best results, put the **break** statement, if any, outside the closing curly brace.

You may also encounter an "**unresolved symbol**" problem at link time if you have link symbols that contain nested types, including nested **enums**. USL C++ Release 2.x, 3.0.x, and 3.0.2.15 encode the names of such symbols differently. The fix is to recompile the problem modules with **cfront** 3.0.2.15. Fortunately, this situation is uncommon.

Supported Windowing Systems

ObjectCenter supports both the Motif and OPEN LOOK windowing systems. Motif is the default on the HP platform. You can choose OPEN LOOK at startup with the **-openlook** switch on the **objectcenter** command line.

ObjectCenter also supports the Common Desktop Environment (found on some HP platforms).

SUN C Library Functions Replaced

To do its run-time error checking and to make its environment behave like a standard UNIX process, ObjectCenter replaces many C library functions and system calls with its own version of the them. For some of these functions, you can substitute your own version. However, ObjectCenter cannot provide run-time error checking on your substituted function.

To use your own version of a function, load the function in a source or object file before linking your program. If your program has already been linked, you must quit, then start a new ObjectCenter session to substitute your function for one of the ObjectCenter replacements.

The following three tables list functions that ObjectCenter replaces. The three tables apply to:

- SunOS (Solaris 1.0) in libc
- Solaris 2 in libc
- Solaris 2 in libucb

Table 1: ObjectCenter replaces these C Library functions on SunOS (Solaris 1.0):

| Name of Function | Can You Substitute Your Own Function? | Name of Function | Can You Substitute Your Own Function? |
|-------------------|---------------------------------------|------------------|---------------------------------------|
| __builtin_alloc() | no | shmdt() | no |
| _exit() | no | shmget() | no |
| _longjmp() | no | shmctl() | no |
| _setjmp() | no | sigaction() | no |
| alloca() | no | sigaddset() | yes |
| bcopy() | yes | sigblock() | no |
| brk() | no | sigdelset() | yes |
| bzero() | yes | sigemptyset() | yes |
| close() | no | sigfillset() | no |
| dup2() | no | sigismember() | yes |
| fork() | no | siglongjmp() | no |
| free() | yes | signal() | no |
| getdtablesize() | no | sigpause() | no |
| getrlimit() | no | sigpending() | no |
| longjmp() | no | sigprocmask() | no |
| malloc() | yes | sigsetjmp() | no |
| mallopt() | yes | sigsetmask() | no |
| mallinfo() | yes | sigstack() | no |
| memalign() | yes | sigsuspend() | no |
| memcpy() | yes | sigvec() | no |
| memccpy() | yes | strcat() | yes |
| memset() | yes | strcmp() | yes |
| mmap() | no | strcpy() | yes |
| munmap() | no | strncat() | yes |
| pause() | no | strncmp() | yes |
| realloc() | yes | strncpy() | yes |
| sbrk() | no | syscall() | no |
| setjmp() | no | system() | yes |
| setrlimit() | no | valloc() | yes |
| shmat() | no | vfork() | no |

Table 2: ObjectCenter replaces these C Library functions on Solaris 2:

| Name of Function | Can You Substitute Your Own Function? | Name of Function | Can You Substitute Your Own Function? |
|-------------------|---------------------------------------|------------------|---------------------------------------|
| __builtin_alloc() | no | shmget() | no |
| _exit() | no | shmctl() | no |
| alloca() | no | sigaction() | no |
| atexit() | no | sigaddset() | yes |
| brk() | no | sigdelset() | yes |
| close() | no | sigemptyset() | yes |
| dup2() | no | sigfillset() | no |
| exit() | no | sigismember() | yes |
| fork() | no | siglongjmp() | no |
| free() | yes | signal() | no |
| getrlimit() | no | sigpause() | no |
| longjmp() | no | sigpending() | no |
| malloc() | yes | sigprocmask() | no |
| memalign() | yes | sigsetjmp() | no |
| memcpy() | yes | sigsuspend() | no |
| memccpy() | yes | strcat() | yes |
| memmove() | no | strcmp() | yes |
| memset() | yes | strcpy() | yes |
| mmap() | no | strncat() | yes |
| munmap() | no | strncmp() | yes |
| pause() | no | strncpy() | yes |
| realloc() | yes | syscall() | no |
| sbrk() | no | sysconf() | no |
| setjmp() | no | system() | yes |
| setrlimit() | no | valloc() | yes |
| shmat() | no | vfork() | no |
| shmdt() | no | | |

Table 3: ObjectCenter replaces these C Library functions on Solaris 2 in libucb.

| Name of Function | Can You Substitute Your Own Function? | Name of Function | Can You Substitute Your Own Function? |
|------------------|---------------------------------------|------------------|---------------------------------------|
| _longjmp() | no | signal() | no |
| _setjmp() | no | sigpause() | no |
| bcopy() | yes | sigsetmask() | no |
| bzero() | yes | sigstack() | no |
| getdtablesize() | no | sigvec() | no |
| getrlimit() | no | syscall() | no |
| longjmp() | no | ucbsignal() | no |
| setjmp() | no | ucbsigpause() | no |
| sigblock() | no | ucbsigblock() | no |

HP C Library Functions Replaced

To do its run-time error checking and to make its environment behave like a standard UNIX process, ObjectCenter replaces many C library functions and system calls with its own version of them. For some of these functions, you can substitute your own version. However, ObjectCenter cannot provide run-time error checking on your substituted function.

To use your own version of a function, load the function in a source or object file before linking your program. If your program has already been linked, you must quit, then start a new ObjectCenter session to substitute your function for one of the ObjectCenter replacements.

The following table lists the C library (libc) functions that ObjectCenter replaces on the HP platform.

| Name of | Can You | Name of | Can You |
|---------|---------|---------|---------|
|---------|---------|---------|---------|

| Function | Substitute Your Own Function? | Function | Substitute Your Own Function? |
|-----------------|-------------------------------------|---------------|-------------------------------------|
| _exit() | no | shmat() | no |
| _longjmp() | no | shmdt() | no |
| _mcount() | no | shmget() | no |
| _setjmp() | no | shmctl() | no |
| alloca() | no | sigaction() | no |
| bcopy() | yes | sigaddset() | no |
| brk() | no | sigblock() | no |
| bzero() | yes | sigdelset() | no |
| close() | no | sigemptyset() | no |
| dup2() | no | sigismember() | no |
| fork() | no | signal() | no |
| free() | yes | sigpause() | no |
| getdtablesize() | no | sigpending() | no |
| getrlimit() | no | sigprocmask() | no |
| longjmp() | no | sigsetmask() | no |
| malloc() | yes | sigstack() | no |
| mallinfo() | yes | sigsuspend() | no |
| mallopt() | yes | sigvec() | no |
| memmove() | yes | sigvector() | no |
| memcpy() | yes | strcat() | yes |
| memccpy() | yes | strcpy() | yes |
| memset() | yes | strncat() | yes |
| pause() | no | strncpy() | yes |
| realloc() | yes | sysconf() | no |
| sbrk() | no | vfork() | no |
| setjmp() | no | | |
| setrlimit() | no | | |

On HP platforms, ObjectCenter replaces the following **libBSD** functions, neither of which can be replaced by the user:

- **signal()**
- **sigvec()**

On HP platforms, ObjectCenter replaces the following **libV3** functions, none of which can be replaced by the user:

- **sigblock()**
- **sigset()**
- **sigpause()**
- **sigrelse()**

SUN Shared Libraries

This section describes the support for shared libraries within the ObjectCenter environment. For general information about Sun shared libraries, see the Sun manual *"Programming Utilities & Libraries"* for SunOS and *"Linker and Libraries Manual - SunOS 5.0"* on Solaris 2.

Reading debugging information on shared libraries is not supported in component debugging mode (CDM, a.k.a. ObjectCenter's Interpreter). Without debugging information on a file, you are unable to perform certain debugging activities, such as stepping through functions. For information about what debugging techniques are possible on code without debugging information, see the **debugging** entry in the **Manual Browser** or *ObjectCenter Reference Manual*.

ObjectCenter supports full source-level debugging of shared libraries in

process debugging mode (PDM). For example, you can step into functions in shared libraries that were compiled with **-g**.

Once you load a shared library into ObjectCenter, its functions and any of its data that you have exported are available to your program. ObjectCenter mimics the behavior of the system's link editors, **ld** and **ld.so**, with regard to loading shared libraries and with regard to binding functions and data.

Search Rules for Loading Libraries

When you load a library using the load command's **-l** switch, ObjectCenter searches for libraries in its search path in the same way that the Sun **ld** command does. ObjectCenter stops searching as soon as it finds either the shared or static version of the library. If it finds both versions in the same directory, ObjectCenter uses the shared version (**.so** file) by default. You can, however, override this default behavior by specifying the binding mode:

- On SunOS (Solaris 1.0):

```
-> load -lX11
Attaching: /s/apps/openwin3.0/lib/libX11.sa.4.3
Attaching: /s/apps/openwin3.0/lib/libX11.so.4.3
-> unload -lX11
Detaching: /s/apps/openwin3.0/lib/libX11.sa.4.3
Detaching: /s/apps/openwin3.0/lib/libX11.so.4.3
-> load -Bstatic -lX11
Attaching: /s/apps/openwin3.0/lib/libX11.a
->
```

- On Solaris 2:

```
-> load -lX11
Attaching: /usr/openwin/lib/libX11.so.4
Attaching: /usr/lib/libsocket.so.1
Attaching: /usr/lib/libnsl.so.1
Attaching: /usr/lib/libintl.so.1
Attaching: /usr/lib/libw.so.1
-> unload -lX11
Detaching: /usr/openwin/lib/libX11.so.4
-> load -Bstatic -lX11
Attaching: /usr/openwin/lib/libX11.a
->
```

Note that ObjectCenter may load more libraries than just the X11 library. When you load a library into ObjectCenter, ObjectCenter also loads any other library referenced by the library you loaded explicitly.

ObjectCenter unloads only the X11 library as a result of the **unload -X11**. The **unload** command unloads only those libraries that you

explicitly unload, regardless of whether ObjectCenter loaded them as a result of a reference or an explicit **load** command.

ObjectCenter allows you to load both the static and shared versions of a library, but we do not recommend that you do so.

On SunOS (Solaris 1), when ObjectCenter loads a shared object (an **.so** file), it looks in the same directory for a data interface description file (an **.sa** file) with the same root name and version number. If it finds such an **.sa** file, it loads it so it can generate the necessary data references. ObjectCenter does not report an error if it fails to find such an **.sa** file.

NOTE: ObjectCenter does not load an **.sa** file with the same root name but different version number as an **.so** file. This situation is treated the same as a missing **.sa** file.

Using Environment Variables to Modify Loading Libraries

One way to affect how ObjectCenter loads shared libraries is by setting environment variables before starting ObjectCenter. Like **ld**, ObjectCenter takes into account the following environment variables.

- **LD_LIBRARY_PATH**

A colon-separated list of directories to search for libraries specified with the **-l** switch. Like **ld**, ObjectCenter looks in directories specified in this environment variable after looking in libraries specified with the **-L** switch on the command line.

- **LD_OPTIONS**

A default set of options to pass to **ld**. The options specified in **LD_OPTIONS** are passed to **load** just as if they were entered first on the command line.

ObjectCenter ignores other environment variables used by **ld**.

NOTE: You must set environment variables before you start ObjectCenter.

Using Switches to Modify the Loading of Libraries

Another way to affect how ObjectCenter loads shared libraries is to use **ld**'s command-line switches with ObjectCenter's **load** command. The following command-line switches affect how ObjectCenter loads libraries:

- **-Bdynamic / -Bstatic**

Specifies binding mode. **-Bdynamic** is the default.

-Bdynamic enables dynamic binding; that is, it uses the shared version of a library if one exists.

-Bstatic forces static binding; that is, it loads the static version of the library.

- **-lx[.v]**

Loads a library with the name **libx.so** or **libx.a**. If **-Bdynamic** is in effect at that point on the command line, loads the latest version of the shared library in the first directory found that contains the library. If no shared version is found, loads the static version.

If you supply a **.v** suffix, only the version specified will be loaded. If that version is not found, ObjectCenter reports an error. If you specify a **.v** suffix to **-l** when **-Bstatic** is in effect, ObjectCenter reports a load-time error.

- **-Ldir**

Adds **dir** to the directories searched for libraries.

ObjectCenter ignores other **ld** command-line switches.

Specifying the Binding Mode

If you want to load the static version of a library instead of the shared version, you can specify the binding mode with the **load** command.

Just as with **ld**, you specify the binding mode using the **-B** switch. The binding mode you specify is in effect until the end of the command line or until you specify another binding mode. Because you can specify binding mode more than once with the **load** command, you can use the shared version of some libraries and the static version of others.

In the following example, the static library **One** (**libOne.a**) is loaded, and the shared library **Two** (**libTwo.so**) is loaded:

```
-> load -Bstatic -lOne -Bdynamic -lTwo
```

Loading the Static Version of the C and C++ Libraries

ObjectCenter automatically loads the C and C++ libraries when starting. By default, it loads the shared version. You can force ObjectCenter to load the static version (**libc.a** and **libC.a**) by setting the environment variable **LD_OPTIONS** to the value **-Bstatic** before starting ObjectCenter. That causes ObjectCenter to load the static versions of all libraries by default. Alternatively, you can unload the shared libraries (using **unload**), then explicitly load the static versions by specifying their pathnames with the **load** command.

When you load a library, ObjectCenter automatically loads any other libraries that the library references. Unloading the first library unloads only that first library, not any of the libraries that were referenced by it. To unload the other libraries, you must unload them explicitly.

Setting Breakpoints in Shared Libraries while in PDM

While you are in **pdm**, you can set a breakpoint in a C library function, such as **printf()**. To do so, you first set a breakpoint in **main()**, issue the **run** command, set the breakpoint in **printf()**, and issue the **cont** command:

```
pdm -> stop in printf  
Function "printf" not defined.  
pdm -> stop in main  
stop (1) set at "main.c:8, main()".  
pdm -> run  
Resetting to top level.  
Executing: /test_dir/a.out
```

```
Breakpoint 1, main (argc=1, argv=0xf7fff824) at  
main.c:8  
pdm (break 1) -> stop in printf  
stop (2) set at 0xf76ed904, printf().  
pdm (break 1) -> cont
```

```
Breakpoint 2, 0xf76ed904 in printf()  
pdm (break 1) ->
```

You have set the breakpoint in this fashion only once; subsequent runs of the program retain the breakpoint in **printf()**:

```
pdm 14 -> run  
Resetting to top level.  
Executing: /test_dir/a.out
```

```
Breakpoint 1, main (argc=1, argv=0xf7fff824) at  
main.c:8  
pdm (break 1) 15 -> cont  
Breakpoint 2, 0xf76ed904 in printf()
```

Binding of Functions and Data

Like the Sun dynamic linker/loader, **ld.so**, ObjectCenter binds functions from shared libraries at run time when the functions are called.

Note that, if you have loaded the definition of a function in your own source or object file, it takes precedence over a definition of that function in a shared library.

On SunOS (Solaris 1), ObjectCenter binds a library's exported initialized data (usually found in the **.sa** file) at link time, not at run time (similar to

ld's statically linking an **.sa** file at link time).

Unloading a Specific Function

If you specify a function in a shared library with **unload**, ObjectCenter unloads the entire **.so** file.

```
-> unload printf  
Detaching: /usr/lib/libc.so.1
```

On SunOS (Solaris 1), ObjectCenter does not unload the **.sa** file.

Other SunOS (Solaris 1) Differences

NOTE: The rest of the section on shared libraries applies **only** to SunOS (Solaris 1).

Unloading Shared Objects

You can unload and unlink an entire shared library at once.

```
-> unload -lX11  
Detaching: /usr/openwin/lib/libX11.sa.4.3  
Detaching: /usr/openwin/lib/libX11.so.4.3
```

Unloading a Specific Module

You can also unload specific modules in a data interface description file (**.sa**) by specifying the module in parentheses following the library name:

```
-> unload /lib/libc.sa.1.6(errlst.o)  
Unloading: /lib/libc.sa.1.6(errlst.o)
```

Using Initialized Global Data

If you declare initialized global data in a program using SunOS (Solaris 1) shared libraries, you should include its initialization in your **.sa** archive, as well as in your **.so** file. If you don't, your program might not use the correct initialization values.

The problem results from behavior of Sun's linker, **ld**, which can only find initializations of data in the user's program or in an archive (**.a** or **.sa** file) - not in **.so** files. When **ld** can't find the initialization for a variable, the system initializes the variable to 0 (zero).

This issue can come up if you use common variables in C programs. Common variables are global variables that are defined in more than one module without using the **extern** qualifier. Consider this simple C example with two files, **a.c** and **b.c**:

```
a.c                                b.c  
int Dogs;                          int Dogs = 12;  
main()  
{printf("Dogs is %d\n", Dogs);}
```

In this example, **Dogs** is a common variable. It is declared in two

modules, namely **a.c** and **b.c**, without using the **extern** qualifier.

Here's what happens when we compile the example statically:

```
% cc -c a.c
% cc -c -pic b.c
% cc -o static a.o b.o
% static
Dogs is 12
```

The program runs fine. But suppose the code from **b.c** is in a shared library without an **.sa** archive. When we link the code, it runs incorrectly:

```
% ld -o libb.so.1.0 b.o -assert pure-text
% cc -o shared1 a.o -L. -lb
% shared1
Dogs is 0
```

The linker erroneously initialized the variable **Dogs** to **0** (zero).

The solution is to create an **.sa** file containing the static data in **b.c**:

```
% ar r libb.sa.1.0 b.o
ar: creating libb.sa.1.0
% ranlib libb.sa.1.0
% cc -o shared2 a.o -L. -lb
% shared2
Dogs is 12
```

Now the program runs correctly. Notice that in this situation you must initialize data in the **.sa** file; otherwise the statically linked program runs differently from the dynamically linked one.

As good practice, you should probably always include initializations in your **.sa** files, even if the static variable is initialized to zero. This results in better shared-library performance.

Using HP Shared Libraries

This section describes the support for shared libraries within the ObjectCenter environment. For general information about HP shared libraries, see the HP Manual, *"Programming on HP-UX"*.

Reading debugging information on shared libraries is not supported in component debugging mode (CDM, a.k.a. ObjectCenter's interpreter). Without debugging information on a file, you are unable to perform certain debugging activities, such as stepping through functions. For information about what debugging techniques are possible on code without debugging information, see the **debugging** entry in the **Manual Browser** or *ObjectCenter Reference Manual*.

ObjectCenter supports source-level debugging of shared libraries in process debugging mode (**pdm**). For example, you can step into functions in shared libraries that were compiled with **-g**.

Once you load a shared library into ObjectCenter, its functions and any of its data that you have exported are available to your program.

ObjectCenter mimics the behavior of the system's link editors in loading shared libraries and binding functions and data, as described below.

Loading Shared Libraries

To load a shared library, use the ObjectCenter **load** command. For example, to load the math library, enter the following:

```
-> load -lm
Attaching: /lib/libm.sl
```

To unload a shared library, use the ObjectCenter **unload** command. For example, to unload the math library, enter the following:

```
-> unload -lm
Detaching: /lib/libm.sl
```

You can also use the **load** and **unload** commands to load and unload libraries with relative or absolute pathnames.

You can check which shared libraries are bound with your executable using the HP **/usr/bin/chatr** command. The output from the **chatr** command has a section headed "**shared library list**". For example:

```
% chatr a.out
a.out:
  shared executable
  shared library dynamic path search:
    SHLIB_PATH disabled second
    embedded path disabled first Not Defined
  shared library list:
    static      a.out
    dynamic     /path/bin/./clcpp/pa-hpux8/a0/libC.sl
    dynamic     /lib/libc.sl
  shared library binding:
    deferred
```

Search Rules for Loading Libraries

When you load a library using the **load** command's **-I** switch, ObjectCenter searches for libraries in its search path in the same way that the HP **ld** command does. ObjectCenter stops searching as soon as it

finds either the shared or archive version of the library. If it finds both versions in the same directory, ObjectCenter uses the shared version (**.sl** file) by default.

You can, however, change the default search path by specifying the **-Ldir** command-line switch. The **-Ldir** switch to **ld** allows you to add additional directories to the search path. If **-Ldir** is specified, **ld** searches the **dir** directory before the default places.

The linker searches libraries in the order in which they are loaded. Libraries specified with the **-l** switch are searched before the libraries that the compiler links in by default. ObjectCenter links in the standard library **libc** and **/lib/milli.a** last.

You can also specifically tell **ld** which libraries to search by setting the **LPTH** environment variable. If **LPTH** is not set, **ld** searches only the libraries specified in **LPTH**; the default libraries are not searched unless they are specified in **LPTH**.

Loading the Archive Version of a Library

If both archive and shared versions of a library reside in the same directory, ObjectCenter loads the shared version by default. You can force ObjectCenter to load the archive version of the library with the **-a search** command-line switch; where **search** can be one of the following: **archive** (use the archive version only), **shared** (use the shared version only), or **default** (use the shared version if available; otherwise use the archive version). If the specified version of the library cannot be found, ObjectCenter reports an error.

The following is an example of using the **-a** archive switch to load an archived version of the math library:

```
-> load -a archive -lm
Attaching: /lib/libm.a
```

The following is an example of using multiple **-a** switches on the same **load** command:

```
-> load -a archive -lm -a shared -lM
Attaching: /lib/libm.a
Attaching: /lib/libM.sl
```

Specifying Global ld Switches with LDOPTS

ObjectCenter supports the HP environment variable, **LDOPTS**, which allows you to specify your **ld** switches in an environment variable. The linker picks up the value of **LDOPTS** and places its contents before any arguments on the command line.

NOTE: You must set the **LDOPTS** and all other environment variables in a shell before starting up ObjCenter in that shell.

Binding of Functions and Data

The default behavior of the HP dynamic linker/loader, **dld.sl**, binds functions from shared libraries at run time when functions are called.

Note that, if you have loaded the definition of a function in your own source or object file, it takes precedence over a definition of that function in a shared library.

ObjCenter binds a library's exported initialized data at link time, not at run time.

Unloading a Specific Function

If you specify a function in a shared library with **unload**, ObjCenter unloads the entire **.sl** file.

Shared Library Stack Frames in Core Files

pdm is currently unable to report about shared library stack frames when statically analyzing core files. Instead, the output will look something like this:

```
Core was generated by 'a.out'.
Program terminated with signal 6, Aborted.
You can't do that without a process to debug
#0 0x800ab0c8 in _end()
pdm 1 -> where
#0 0x800ab0c8 in _end()
#1 0x800ab098 in _end()
```

In order to get an accurate stack trace, reproduce the fault within **pdm**.

```
pdm 1 -> run
Resetting to top level.
Executing: a.out
```

```
Reading symbols from /lib/libc.sl...no debugging symbols
found...done.
```

```
Program received signal SIGABRT, Aborted.
0x7b0110c8 in kill ()
pdm (break 1) 2 -> where
#0 0x7b0110c8 in kill ()
#1 0x7b011098 in export stub
#2 0x7aff3368 in raise ()
#3 0x7aff32e4 in export stub
#4 0x7aff2f18 in abort ()
#5 0x7aff2d34 in export stub
```

```

#6 0x1e74 in foo(const void *, const void *)
(a=0x0, b=0xa "") at h.c:5
#7 0x1e10 in export stub
#8 0x7afe3a84 in _qsort ()
#9 0x7afe39ac in _qsort ()
#10 0x7afe38d4 in export stub
#11 0x1ed4 in main ()

```

**Link with
libCdynamic to Invoke
Static Initializers**

The CenterLine C++ translator distributed with ObjectCenter 2.2.0 invokes static/global constructors and destructors from shared libraries when linked with a dynamic version of **libC**. To link with the dynamic library, use the **-set_lib_id** switch on the **CC** command line:

```
% CC -set_lib_id=Cdynamic file.c
```

Alternatively, you can set the **LIB_ID** environment variable to **Cdynamic** before compiling with **CC**:

```
% setenv LIB_ID Cdynamic
% CC file.c
```

**Calls to shl_* (3X)
Functions**

This section documents the behavior of **shl_*(3X)** functions when you call them within ObjectCenter. These functions are available only on the HP 9000 Series 700 and Series 800 platforms.

shl_definesym(3X) always returns **-1** and sets **errno** to **EINVAL**. ObjectCenter is unable to accurately reproduce the semantics of **shl_definesym(3X)**, so it is not supported.

shl_load(3X) and **cxxshl_load(3X)** always interpret the **BIND_IMMEDIATE** modifier flag as if it were **BIND_DEFERRED**. It is impossible to obtain **BIND_IMMEDIATE** semantics from **shl_load(3X)** and **cxxshl_load(3X)** within the ObjectCenter environment.

Loading a library with the **BIND_FIRST** modifier flag to **shl_load(3X)** or **cxxshl_load(3X)** will cause that library to be assigned an index of 1 (see the **shl_get(3X)** manual pages for a description of a library's index). By contrast, loading a library outside of the ObjectCenter environment will cause that library to be assigned an index of 0 (zero). Therefore, an index of 0 (zero) always represents the user's main program within the ObjectCenter environment.

When the **DYNAMIC_PATH** flag is passed to **shl_load(3X)** or **cxxshl_load(3X)**, those functions use only the path list stored in the **SHLIB_PATH** environment variable to find the library. This is because there is no way to emulate the **+s** and **+b** options to **ld(1)** within the

ObjectCenter environment.

ObjectCenter unloads all dynamically loaded libraries when a reset to top level occurs.

shl_unload(3X) and **cxxshl_unload(3X)** do not actually unload the library, and thus do not unbind any symbols that have been bound from that library. Unloading and relocating a library with **shl_*(3X)** functions will not cause them to return an error status, but the original image of the library is the only one that is ever loaded during a single run of the user program.

The value of the HP-UX reserved variable, **__dld_loc** is always **0x0** within ObjectCenter. It should not be used by a user program.

C++ static constructors and destructors in shared libraries are always invoked, even if the user program does not use the C++-aware dynamic loading functions, **cxxshl_load(3X)** and **cxxshl_unload(3X)**.

ObjectCenter does not call initializer functions in a shared library at any time. Note that initializer functions in HP-UX shared libraries are not C++ static initializers, which ObjectCenter supports. Initializer functions are language-independent functions defined at shared-library-creation time with the **+I** option to **ld(1)**.

The initializer member of **struct shl_descriptor** objects generated by **shl_get(3X)** and **shl_gethandle(3X)** always have the following value: **NO_INITIALIZER**.

shl_get(3X) will return -1 (with **errno == EINVAL**) if it is passed an index with the value -1 or 0 (zero). This is because there is no distinct dynamic linker (index -1) or main program executable (index 0) within the ObjectCenter environment.

shl_gethandle(3X) will return -1 (with **errno == EINVAL**) if it is passed the handle **PROG_HANDLE**. This is because there is no distinct main program executable (index 0) within the ObjectCenter environment.

shl_getsymbols(3X) behaves as if every defining instance of a symbol with external linkage in the user's program is exported from the main program. These symbols are accessed by passing **PROG_HANDLE** as the first argument to **shl_getsymbols(3X)**. Outside of the ObjectCenter environment, this behavior is achieved by using the **-E** switch to **ld(1)** when creating the main program.

The **GLOBAL_VALUES** and **IMPORT_SYMBOLS** modifier flags are not supported for calls to **shl_getsymbols(3X)**. Calls to **shl_getsymbols(3X)** will return -1 with **errno** set to **EINVAL** if either of these flags is passed in.

Potential SUN Anomalies

In most cases, your programs will run the same within the ObjectCenter environment as they do outside the environment. However, there are some platform-specific features that ObjectCenter may not support fully, so you may see unexpected behavior.

This section attempts to call your attention to these potential anomalies on Sun platforms. Unless otherwise specified, these anomalies apply to both SunOS (Solaris 1) and Solaris 2.

Default Parser Configuration

The default C parser configuration for the ObjectCenter interpreter (CDM) is K&R.

Invoke the **config_c_parser** command to display the current setting. For more information about specifying a different parser configuration, see the **config_c_parser** entry in the *ObjectCenter Reference*.

Sun C Compilers and pdm

On Solaris 2, when you try to debug an executable compiled with SunPro C compiler Version 2.0.1 (or higher) or with SPARCompiler C Version 3.0.1 (or higher), **pdm** may report that no debugging symbols were found. This is because the Sun compiler puts the debugging information that **pdm** uses in the **.stab.excl** section of the executable, which is stripped by the static linker during final execution.

To work around this problem, compile with the **-xs** switch. This causes the assembler to place debugging information in the **.stabs** section, which is not stripped by the linker.

Switches and Variables Ignored by load (SunOS/Solaris 1)

In SunOS (Solaris 1), ObjectCenter's **load** command ignores some command-line switches that are accepted by Sun **cc**, but many of these switches do not change the meaning of a program. The following Sun **cc** command-line switches ignored by ObjectCenter do change the meaning of a program.

- **-align_block**
- **-fnonstd**
- **-ffloat_option**
- **-misalign**

The **load** command also ignores the **FLOAT_OPTION** environment variable that is accepted by Sun **cc**, which does change the meaning of a program.

ObjectCenter does not support object files created using the Sun **cc** compiler **-dalign** switch. This switch causes more stringent alignment

rules for storing double-precision values in memory. Calling object code functions compiled with **-dalign** from source code could result in a segmentation violation. However, you can load your whole application as object code without problems.

Instrumented Object Code

On Sun workstations, ObjectCenter fails to generate warnings for bad pointer dereferences inside **switch** statements in instrumented object code compiled without debugging information. You can avoid this limitation by compiling with the **-g** switch to add debugging information.

Setting LD_LIBRARY_PATH

To use the ObjectCenter tutorial on Solaris 2 platforms, you must set the following environment variables:

```
setenv OPENWINHOME /usr/openwin
setenv LD_LIBRARY_PATH \
/usr/openwin/lib:/usr/lib:$LD_LIBRARY_PATH
```

On Solaris 2, the linker only checks for major numbers. Some components of ObjectCenter, for example the **vi** integration, require the library **libX11.so.4**. If ObjectCenter issues an error message such as "**cannot find libX11.so.4**", make sure that **/usr/openwin/lib** contains a symbolic link called **libX11.so.4** pointing to a **libX11.so.4.x** or **libX11.so.5.x** library.

Locating X11 Header Files

The tutorial assumes the X11 header files are installed in **/usr/include**. If they are not, contact your system administrator to put a copy or symbolic link to the location of the X11 header files into **/usr/include**, or add **-Ipathname** to the **CL_INCS** link in the tutorial **Makefile**, where **pathname** is the path to the directory containing the X11 header files.

If you use the **X11R5** libraries instead of the **openwin** libraries, you must explicitly load **-lnsl** and **-lsocket** into the Workspace to run the tutorial. These dependencies are not automatically included in the **X11R5** libraries, whereas they are included in the **openwin** libraries.

Resource Limits for Stacksize

If a user has set the **stacksize** resource to **unlimited**, it may affect ObjectCenter's **pdm** when debugging an executable. At run time, if the application being debugged within **pdm** experiences a run time error (such as a segmentation fault), ObjectCenter may not display any relevant error information in the Workspace. Essentially, no source file information is displayed, including the line of code where ObjectCenter stopped execution and where the problem may be occurring. The user will only understand that a problem has occurred and that execution of the program has stopped but not have much to go on to resolve the

problem.

Using the ObjectCenter tutorial as an example, we can see how this problem takes effect. After running the **bounce_dump** application in ObjectCenter's **pdm**, the user may see the following output if the **stacksize** resource has been set to **unlimited**:

```
pdm -> debug bounce_dump core  
Debugging program 'bounce_dump'  
Core was generated by 'bounce_dump'.  
Program terminated with signal 10, Bus error.  
#0 0x10df8 in store_shape ()  
pdm ->
```

As you can see, the user would only understand that a runtime error had occurred in the function called 'store_shape', but not the line number at which the problem occurred. This is a problem that occurs on both SunOS (Solaris 1) and Solaris 2 systems. The error resulting may be slightly different, but the problem is essentially the same.

To avoid this problem, quit out of ObjectCenter and then at the Unix prompt set the "soft limit" of the **stacksize** resource to **2048** as follows:

```
% limit stacksize 2048
```

Then restart the **pdm** session. The user should now obtain the proper run time error information they had originally expected.

Going back to the ObjectCenter tutorial example, the above messages change to the following once the **stacksize** resource limit is changed to **2048**:

```
pdm -> debug bounce_dump core  
Debugging program 'bounce_dump'  
Core was generated by 'bounce_dump'.  
Program terminated with signal 10, Bus error.  
#0 0x10df8 in store_shape (count=0, shape=0xefff680  
"rectangle") at shape.c:11  
11 *old = *new;  
pdm ->
```

You can see how the output changes to what is depicted within the ObjectCenter tutorial manual, giving the user more details on the problem. Additionally, at run time of an application, the **Source Area** is able to display the source code and a pointer to the line number where execution stopped and where the error is occurring.

There is no explanation at this time as to why the **stacksize** resource and the soft limit setting established for this resource affects ObjectCenter.

However, if this resource is set to **unlimited** or pretty much anything higher than **2048**, the problem described above could occur.

To find out more about "soft limits" and the other resources that can have a limitation placed on them, see the **man** pages for the **limit** (or **ulimit**) command or talk to your system administrator, who can adjust the system resource limitations for your login account as necessary.

Potential HP Anomalies

In most cases, your programs will run the same within the ObjectCenter environment as they do outside the environment. However, there are some platform-specific features that ObjectCenter may not support fully, so you may see unexpected behavior. This section attempts to call your attention to these potential anomalies.

Default Parser Configuration

The default C parser configuration for the ObjectCenter interpreter is K&R.

Invoke the **config_c_parser** command to display the current setting. For more information about specifying a different parser configuration, see the **config_c_parser** entry in the *ObjectCenter Reference*.

Supported cc and c89 Switches

ObjectCenter's **load** command supports the following switches to the HP **c89(1)** and **cc(1)** commands:

- **-Dname=def** or **-Dname**

to define a preprocessor symbol

- **-lx**

to cause the linker to search the library **libx.a** or **libx.sl**, where **x** is one or more characters, in an attempt to resolve currently unresolved global references.

- **-Ldir**

to tell the linker to search in **dir** for **libx.a** or **libx.sl** before searching in the default locations.

- **-Uname**

to remove any initial definition of **name** in the preprocessor.

- **-w**

to suppress load-time warning messages.

Supported ld Switches ObjectCenter's **load** command supports the following switches to the HP **ld(1)** command:

- **-a [archive | shared | default]**

to specify whether shared or archived libraries are searched with the **-l** switch. By default, the shared version is used if it exists, and the static version if it does not. If **-a archive** or **-a shared** is specified, only the specified library type is accepted.

- **-lx**

to cause the linker to search the library **libx.a** or **libx.sl**, where **x** is one or more characters, in an attempt to resolve currently unresolved global references.

- **-Ldir**

to tell the linker to search in **dir** for **libx.a** or **libx.sl** before searching in the default locations.

The CenterLine C++ translation system also supports the following switches to the HP **ld(1)** command:

- **+b path_list**

to specify a colon-separated list of directories to be searched at run time. **path_list** is searched for shared libraries needed by the executable that were specified with **-l**. If **path_list** is a single colon (;), **ld** builds the list from all the directories specified by **-L** and in the **SHLIB_PATH** environment variable.

- **+s**

to specify that shared libraries can be located using the environment variable **SHLIB_PATH**. **SHLIB_PATH** should be set to a colon-separated list of directories. If **+s** and **+b** both appear on the command line, the path specified by the switch which appears first is searched first.

NOTE: the **-a** switch to the CenterLine **CC** command is not position-dependent, that is, it can be used only once on a command line. If more than one **-a** switch is issued, the

rightmost **-a** switch takes precedence.

Different Behavior of the `load` Command

ObjectCenter processes all **-L** switches before processing any **-I** switches. Thus, the command:

-> `load -lm -L./subdir`

loads the library `./subdir/libm.a` instead of `/lib/libm.a` (presuming both libraries exist). HP's **ld(1)** command processes all command-line switches in left-to-right order, so this behavior is unexpected.

Unsupported Environment Variables

Currently, ObjectCenter's **load** command ignores these HP-UX environment variables:

- **CCOPTS**
- **FLOW_DATA_DIR**
- **LANG**
- **LPATH**
- **LD_PXDB**

No PIC Support

The ObjectCenter dynamic linker/loader does not support object modules containing Position Independent Code (PIC) on the HP platform.

No Support for Cache Hint Bits

The ObjectCenter dynamic linker/loader does not detect and eliminate use of cache hint bits in object modules in situations where 64-byte stack pointer alignment cannot be guaranteed.

void Types may be Misinterpreted

Due to an HP compiler bug, objects of base type **void** can sometimes appear to have the base type of **int** when their definitions are loaded in object form.

No Support for Long Pointers

The ObjectCenter interpreter does not support long pointer declaration syntax (for example, "**int ^x;**").

No Support for Argument Values in Ininsics

The ObjectCenter interpreter does not support default argument values in calls to intrinsics.

ObjectCenter Replaces Fewer Header Files

We provide fewer header files on the HP platform than we do for some other platforms because more HP header files comply with the ANSI standard.

Changes in Signal Handling

If your program makes use of either **signal()** in **libBSD.a** or **sigpause()** in **libV3.a**, load the appropriate library before the program is linked or run for the first time in a session; otherwise, the **libc.a** version will be used.

After a program is linked or run for the first time in a session, you cannot subsequently change the version of **signal()** or **sigpause()** that your program uses. Therefore, if you link or run without first loading the appropriate library, you must exit ObjectCenter and restart it.

To avoid this problem, add one of the following lines to your personal **.ocenterinit** file or to the system-wide **ocenterinit** file:

- For **libBSD.a**, add:

load -lBSD

- For **libV3.a**, add:

load -lV3

sigvec() and **sigvector()** Flags

The **SV_ONSTACK** and **SV_NOCLDSTOP** flags to **sigvec()** and **sigvector()** are ignored. (This behavior with **SV_ONSTACK** is the same as on other architectures supported by ObjectCenter; however, the behavior with **SV_NOCLDSTOP** is specific to HP-UX).

The **SV_BSDSIG** flag used in **sigvector()** has no effect (that is, the **sigvector()** function always acts as if the **SV_BSDSIG** flag is set). The **SV_BSDSIG** flag is not returned in the old **sigvec** structure, even if it was set by the caller.

sigaction() Flags Ignored

The **SA_ONSTACK** and **SA_NOCLDSTOP** flags to **sigaction()** are ignored.

No **sigspace()** and No **sigstack()**

The **sigspace()** and **sigstack()** functions are not supported.

_longjmp() and **_setjmp()**

The **_longjmp()** and **_setjmp()** functions behave like **longjmp()** and **setjmp()**.

Enhanced **sigset()** and **sigvector()**

Signal handler maintenance functions which are incompatible under HP-UX (such as **sigset()** and **sigvector()**) work correctly with each other under ObjectCenter.

syscall() Function

The function **syscall()** is unsupported in HP-UX 9.xx and possibly HP-

UX 10.x. Calling this unsupported function can result in unexpected behavior.

ioctl() Support

ObjectCenter supports user programs that make **IPMAPMAP** and **IPMAPUNMAP ioctl()** requests. There is one restriction.

The restriction is that ObjectCenter considers a memory area invalid if your program maps it with **IPMAPMAP** and unmaps it with **close()** or **ioctl()**. This is the case even if **close()** or **ioctl()** did not actually unmap the area because other processes on the system still had it mapped. A subsequent **IOMAPMAP ioctl()** request for that memory area causes ObjectCenter to again consider it valid.

Static Symbol not in symtab

On the HP platform, **pdm** may issue the following warning:

**Internal: static symbol 'symname' found in filename
psymtab but not in symtab**

If you receive this message, issue the following command:

whatis <symname>

and then reissue the command that caused the warning.

Attaching to a Running Process may Fail on HP-UX

Attaching to a running process sometimes fails on HP-UX in ObjectCenter's process debugging mode (**pdm**) if the **pdm** binary you are using is installed on a remote partition.

The failure looks like a **ptrace** failure, such as one you get if you request an illegal process number or if you attempt to attach a file you do not own.

```
pdm -> debug a.out  
process_id debug: ptrace: Permission denied.  
pdm ->
```

Before implementing the workaround we provide below, eliminate any possibility that the failure is a result of a **ptrace** error. See the **ptrace man** page for information about **ptrace** failure errors.

This is the workaround:

- Copy the **pdm** binary to a disk that is local to the HP-UX machine.

You must copy it on a disk local to each HP-UX machine one

which you want **pdm** to run. Do not install it on a cluster partition.

- Create a **.clpm.conf** file in your home directory to override the **AS PDM** directive in the system **.clpm.conf**.

If you have multiple home directories, you have to create the multiple **~/.clpm.conf** files so that the workaround affects each machine on which you want **pdm** to run.

The **.clpm.conf** file should contain the following lines:

```
AS PDM {
  BINARY: /tmp/pdm;
  BINARY_ENV: CLDB;
  ENV:
  LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${centerline}/${arch}/lib;
  ARGS: -connect;
  RESTART_ARGS: -connect -restart;
  ARG_TMPL: ^[^-].*:0;
}
```

With the workaround, attempts to attach to a running process will now succeed:

```
pdm 1 -> debug a.out pid
Debugging program '/net/pickup/tmp/loop'
Resetting to top level.
warning: reading register r4: I/O error
0x2560 in sigpause()
pdm (break 1) 2 -> where
#0 0x2560 in sigpause ()
#1 0x21cc in _sleep ()
#2 0x1fec in main () at loop.c:16
```

Attaching to a process may also fail if the process is sleeping. **pdm** cannot attach to a sleeping process. For more information about attaching to a running process, see the **debug** entry in the **Manual Browser** or *ObjectCenter Reference*.

clcc Command-Line Switches

Many of the switches used with the HP C compiler are also used with **clcc**, the CenterLine-C compiler. However, some switches you use with **cc** must be replaced by a corresponding **clcc** switch, and other **cc** switches have no equivalent in **clcc**.

The following table shows some common **cc** switches and their **clcc** equivalents.

| HP cc Switch | clcc Switch | Description |
|--------------|------------------------------|--|
| -Aa | -ansi | Enables strict ANSI compliance. |
| -Ac | -traditional, -Xa, or -Xt | Disables strict ANSI C compliance. Please refer to the descriptions of |

| | | |
|------------------------------|-----------------------------------|---|
| | | <p>these switches in the <i>CenterLine-C Programmer's Guide</i> or the clcc manual page for more information.</p> |
| -G | -pg | <p>Inserts information required by the gprof profiler in the object file.</p> |
| -Wx,arg1[,arg2, ...,argn] | -Hcppopt=string -Hldopt=string | <p>Pass argument string to preprocessor (-Hcppopt) or the linker (-Hldopt). 'string' can contain multiple arguments separated by commas.</p> |
| +L | -Hlist | <p>Generates a source listing on standard output.</p> |
| +On | -On | <p>Sets the optimization level to 'n' where 'n' is 1 to 7. If 'n' is not specified, then a level of 2 is used.</p> |
| +wn | -wn | <p>Suppress warning messages at level 'n' and higher where 'n' is 1 to 4.</p> |
| +z | -pic | <p>Produces position- independent code. The memory allocated to static variables cannot exceed 4K.</p> |
| +Z | -PIC | <p>Like -pic, but allows the global offset table to span the range of 32-bit addresses when there are too many global objects for -pic.</p> |