Soft Externals [10]

This section describes the special handling that the loader performs when processing "soft" references to an external symbol, or "soft externals."

Soft external references

10.1

Soft externals let the user control whether modules containing entry points to external functions or data objects are linked to the user's program. If the user program declares a reference to an external function as "soft," that reference is not sufficient to ensure that the external function will be included in the program. The function will be included only when referenced elsewhere in the program.

For example, Figure 11 contains two user programs, *flowpgm* and *noflwpgm*. *flowpgm* calls `flowtrace`, performs several functions, and then calls `exit`. *noflwpgm* does *not* call `flowtrace`, but it performs several functions, and then calls `exit`. The `exit` routine is called by both user programs; it processes `exit` calls for programs that call `flowtrace` and for programs that do *not* call `flowtrace`. Therefore, `exit` contains conditional calls to `flowexit`, which is an entry point within the `flowtrace` module. If `flowexit` is declared as a "hard," or normal external reference in `exit`, all of the `flowtrace` module must be loaded with each user program that calls `exit`, regardless of whether the user program calls `flowtrace`. If `flowexit` is declared as a soft external, then the `flowtrace` module is linked to the user program only when `flowtrace` is referenced. In Figure 11, the `flowtrace` module will be loaded with *flowpgm*, but it will not be loaded with *noflwpgm*.

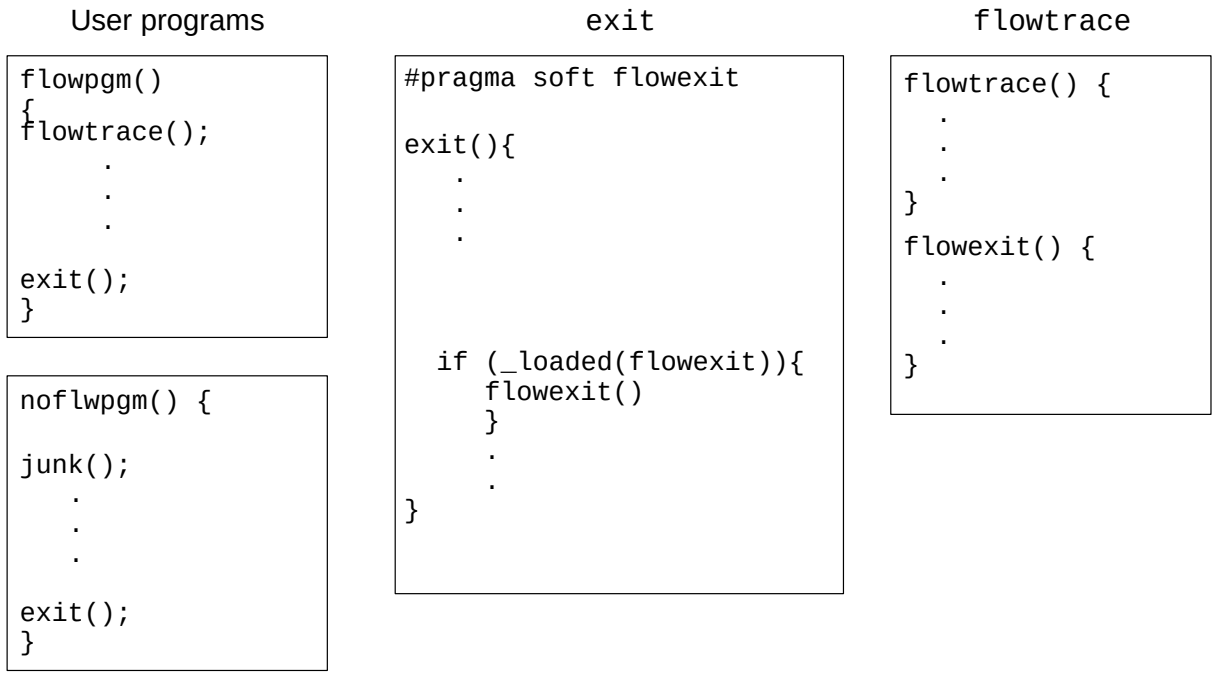

Figure 11. Soft external usage

How to declare soft externals

10.2

References made to entry points located outside a compilation unit are usually “hard,” or normal references. The assembler (`as`(1)) and C compiler (Cray C compiler version 5.0 and on and Cray Standard C compiler version 2.0 and on) allow you to declare a reference to be soft.

A soft external in assembly language is declared by using the `soft` modifier on the `ext` directive. For example:

```
ext getmsg:soft
```

This statement declares that all references in this module to the external symbol `getmsg` will be soft references.

To declare a soft external in C, use the `#pragma` directive, as follows:

```
#pragma soft getmsg
extern int getmsg();
```

The `#pragma` directive should appear before any references to the external entry point. The directive affects the entire source file.

How to link soft externals

10.3

The loader handles hard and soft references in different ways. If the definition has been found by the loader, hard references to an external entry point are always satisfied by the symbol definition. A hard reference to a library entry point will cause the module containing that entry point to be included in the executable program.

A soft reference is not automatically satisfied by the symbol definition. To satisfy the soft reference, the entry point must be included in the program for some other reason. A soft reference to a library entry point is not sufficient to cause the module containing that entry point to be included in the executable program.

You can cause the library entry point to be included in the program by including one of the following in your program:

- Include hard references to the entry point in the program.
- Include hard references to other entry points in the same module so that the module will be included in the program.
- Force-load the object module. See “Including object modules,” page 24, for a discussion of object module inclusion and force-loading.

As is the case with hard references, if the entry point is included in the program, the soft reference is satisfied by the entry point. If the entry point is not included in the program, the soft reference is converted into an unsatisfied external reference. If the reference has not been satisfied, no error message will be generated indicating that the reference is unsatisfied. If the entry point is referenced during program execution, an appropriate error message will be issued and program execution will terminate.

Using soft externals
10.4

At load time, the loader determines if a soft reference should be linked to the corresponding entry point. An execution-time test is needed to determine whether the reference is satisfied and can be called. You can either use the library routine `_loaded`, or use a flag word, to perform the test.

Testing entry-point references with `_loaded`
10.4.1

If the input argument to the library routine `_loaded` is an entry point that has been included in the program, the library routine `_loaded` returns a nonzero value.

The following example is a simplified version of the program exit processing, and it illustrates the use of `_loaded`. The `exit` routine is called at the end of every program. It needs to call the `flowexit` routine if flowtrace processing has been enabled; `flowexit` is contained in the same module as the entry point `flowtrace`. The `flowtrace` entry point will be called if the flowtrace processing is enabled; therefore the soft reference to `flowexit` from `exit` will be satisfied. If `flowtrace` is not called, the soft reference to `flowexit` from `exit` will not be satisfied. The code in `exit.c` that calls `flowexit` takes the following form:

```
#pragma soft flowexit
extern int flowexit();
extern int _loaded();
exit () {
...
  if (_loaded(flowexit))
     flowexit();
}
```

Testing entry-point references with flag words
10.4.2

The second test method uses a flag word rather than the `_loaded` routine. The following code uses the same example to illustrate how a flag word is used:

```
/* flowtrace.c */

int flowflag = 1;
flowtrace () {
 ...
}
flowexit () {
 ...
}


/* exit.c */

int flowflag;
exit () {
...
if (flowflag)
  flowexit();
}
```

If the module from `flowtrace.c` is included, `flowflag` will have a value of 1, and `flowexit` will be called. If `flowtrace.c` is not included, `flowflag` will be 0 and `flowexit` will not be called.

How to convert soft references to hard references
10.5

The HARDREF loader directive can be used to force the loader to treat all soft references to one or more entry points as hard references. The loader treats all soft references to the specialized entry points as hard references, and it will satisfy the reference if the definition is found. You can use the HARDREF directive to force the satisfaction of a reference even when no other condition would cause it to be satisfied.

HARDREF *directive*
10.5.1

The HARDREF directive specifies one or more entry points that should be included in the load process. Any soft references made to these entry points are converted into hard references.

Format:

```
HARDREF=epname1[,epname2...]
```

epname$_i$ Name of entry point from which all soft references will be converted to hard references.

How to convert hard references to soft references
10.6

The SOFTREF directive can be used to force the loader to treat all hard references to one or more entry points as soft references. The loader treats all hard references to a symbol name as soft references. The module containing the indicated entry point is included in the program only when some other factor causes the inclusion. (See subsection "How to link soft externals," page 105, for information.)

SOFTREF *directive*
10.6.1

The SOFTREF directive specifies one or more entry points that should not be included in the load process.

The SOFTREF directive should be used with caution, because it can cause references to symbols to remain unsatisfied, for which no loader error message will be issued. If a program does not make a run-time test to determine whether the reference has been satisfied, and the reference is executed at run time, the program terminates in error.

Format:

```
SOFTREF=epname1[,epname2...]
```

epname$_i$ Name of entry point from which all hard references will be converted to soft references.