Fortran Signal Handling

Ge Baolai
SHARCNET
The University of Western Ontario

June 19, 2008

1 Signal Handling in User Applications

A running programme may be interrupted or terminated by the operating system. When an interruption occurs, a signal is delivered to the process. The running process may take appropriate actions once it receives a signal. This tutorial describes how to handle signals issued by the system during the execution of a Fortran programme.

Signal handling in C is done through the call to system function `signal(2)`. The call

`signal(`*signum*`,` *handler*`)`

installs a user defined routine *handler*, which will be invoked at a later time when the user selected signal *signum* arrives. When the user routine *handler* is called, it takes the actions defined by the user. For instance, the following call

```
signal(SIGTERM, action_sigterm)
```

installs a user defined function `action_sigterm()`, which will be invoked when the process catches `SIGTERM`. When the operating system kills a process, it sends a `SIGTERM`, followed by a `SIGKILL`. The handler routine, when invoked, gives the control back to the process and a (last) chance to take actions, such as writing current data to disk files.

Designed solely for numerical computation, the Fortran standard, until the late 1990s, did not have intrisic procedures or APIs defined for signal handling. Instead exceptions, primarily of interest to floating point arithmetics such as division by zero, illegal arguments to mathematical functions, etc. are handled by runtime libraries. Upon the occurrance of such exceptions, errors are reported and the process is terminated, unless default actions are predefined, mostly at compile time.

While handling exceptions by runtime libraries greatly minimizes the programming efforts of Fortran programmers, there are situations that being able to deal with the exceptions by users before the programme quits abnormally appears to be valuable. For instance, in the event that a programme is being

terminated by the system when a scheduled runtime has been exhausted, one woule like to gracefully shutdown the programme by saving the states of variables and intermedium results from the computation before the programme is killed. In order to gracefully shutdown a programme, the application must be able to first catch the signal and then take necessary actions accordingly.

2 Signal Handling in Fortran as An Extension

Some vendor supplied Fortran implementations, including for example digital, IBM, Sun and Intel, had the extension that allows the user to do signal handling as in C. The interface for installing a signal handler appears the same

```
call signal(signum, handler)
```

where *signum* is the value of signal defined for the targetted architecture, as shown in Figure 1, and *handler* is a user defined procedure in the form of subroutine. The signal numbers are architecture dependent. Figure 2 shows

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
```

Figure 1: Symbolic names and values of common signals as returned from command `kill -l` for Linux i686.

a different set of values defined on Linux Alpha. In order for the code to be

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGEMT       8) SIGFPE
 9) SIGKILL     10) SIGBUS      11) SIGSEGV     12) SIGSYS
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGURG
17) SIGSTOP     18) SIGTSTP     19) SIGCONT     20) SIGCHLD
21) SIGTTIN     22) SIGTTOU     23) SIGIO       24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGINFO     30) SIGUSR1     31) SIGUSR2     32) SIGRTMIN
```

Figure 2: Symbolic names and values of common signals as returned from command `kill -l` on Linux Alpha.

portable, one should use the symbolic names instead of values.

The interface `signal()` requires that parameters that are to be passed to the handler, if any, would have to be visiable to the handler as globals, except for *signum*, which is the only argument that can be passed the the handler as a dummy argument.

GNU Fortran has defined instrinsic `signal()` as an extension

`call signal(`*signum*`, `*handler*`[, `*status*`])`

where the third argument, which is optional featured by the support of polymorphism in Fortran 90 standard, stores the return value of the call to system function `signal(2)`.

The behaviour of the Fortran extention of `signal()` is implementation dependent. In the following example,

```
program fsignal_test
   ... ...
   external warning_sigint ! Must declare as external

   call signal(SIGINT, warning_sigint)
   call sleep(30)
end program

subroutine warning_sigint
   print *, 'Process interrupted (SIGINT), exiting...'
   return
end subroutine warning_sigint
```

compiled using Intel Fortran compiler 10.0, when the programme is interrupted from the command line by `Ctrl+C` key stroke, the handler `warning_sigint` is not invoked. Instead, one will see the following

```
forrtl: error (69): process interrupted (SIGINT)
Image              PC        Routine            Line        Source
a.out              0808EBAF  Unknown               Unknown  Unknown
a.out              0808E1CF  Unknown               Unknown  Unknown
a.out              0806B66A  Unknown               Unknown  Unknown
a.out              0805DAB8  Unknown               Unknown  Unknown
a.out              0804A3FD  Unknown               Unknown  Unknown
.                  002FD420  Unknown               Unknown  Unknown
a.out              08049D2C  Unknown               Unknown  Unknown
libc.so.6          00469F70  Unknown               Unknown  Unknown
a.out              08049AC1  Unknown               Unknown  Unknown
```

The signal `SIGINT` is intercepted by the Fortran runtime library. In order to catch the signal `SIGINT` and take actions defined in the handler `warning_sigint`, one needs to add the following C code

```
void sigclear_(int *signum)
```

```
{
    signal(*signum, NULL);
}
```

and call it in the Fortran code before the installation of signal handler

```
program fsignal_test
   ... ...
   external warning_sigint ! Must declare as external

   call sigclear_(SIGINT)
   call signal(SIGINT, warning_sigint)
   call sleep(30)
end program
```

Technical Points A set of exception handling intrinsic functions have been introduced to Fortran 95 and later standards. Interested readers are recommended to read John Reid's historical notes [2] and the book by the same author [1], as well as and the documentations of the latest Fortran standards concerning exception handling of IEEE floating point arithmetics.

---

3 A User Level Approach

The support for signal handling can be achieved at user level as well with minimum efforts involved. Assume we want to have the same interface as supported as an extension in some Fortran flavours, i.e.

```
call signal(SIGTERM, action_sigterm)
```

We need to write a C routine that makes a call to system function `signal()`. The C code, stored in a separate file `csigfun.c` will look as simple as the following:

```
/* in "csigfun.c" */
#include <signal.h>

typedef void (*sighandler_t)(int);

void signal_( int* signum, sighandler_t handler)
{
    signal(*signum, handler);
}
```

The follwing Fortran code shows a simple example of calling the C function `signal()` to install signal handlers

```
! in "fsignal_test.f90"
program fsignal_test
   ... ...
   external warning_sigterm ! Must declare as external
   external warning_sigint ! Must declare as external

   ! Install signal handlers, return immediately
   call signal(SIGTERM, warning_sigterm)
   call signal(SIGINT, warning_sigint)

   ! Do something that will take some time
   call sleep(30)
end program

subroutine warning_sigterm
   print *, 'Process interrupted (SIGTERM), exiting...'
   return
end subroutine warning_sigterm

subroutine warning_sigint
   print *, 'Process interrupted (SIGINT), exiting...'
   return
end subroutine warning_sigint
```

Each call to the C routine `signal()` installs a signal handler for the specified *signum* (`SIGTERM` and `SIGINT`), which is defined as a subroutine in the same file. Note that the call to `signal()` is nonblocking. That is, it returns immediately, thus the programmem continutes to execute the subsequent instructions.

We show in the following how to compile and run the test programme that contains parts written in mixed Fortran and C languages. Without loss of generality, we assume the name for the C compiler is `cc` and the name for the Fortran compiler is `fort` We compile the C code first to obtain an object file using command

```
cc -c csigfun.c
```

This will create a object file named `csigfun.o`. Then we compile the Fortran code, and link with the C object

```
fort fsignal_test.f90 csigfun.o -o fsignal_test
```

to generate the executable `fsignal_test`.

Start the executeable from command line

```
./fsignal_test
```

Note that the execution of the call to `sleep(30)` will put the programme in sleep mode for about 30 second, which gives us enough time to open another terminal to find the process ID and issue an signal from command line.

```
[bge@mobile-hpc]$ ps -ef | grep a.out
bge       4314  4282  0 20:47 pts/1    00:00:00 fsignal_test
bge       4316  3094  0 20:47 pts/0    00:00:00 grep fsignal_test
[bge@mobile-hpc]
[bge@mobile-hpc]$ kill -s INT 4314
```

As soon as the interrupt signal `INT` is issued from the command line, in the terminal from which we started the programme, the signal handler `warning_sigint()` is invoked and a message from the subroutine `warning_sigint()` is printed and the programme then exits.

```
[bge@mobile-hpc]$
 Process interrupted (SIGINT), exiting...
```

The source code of Fortran signal support and the example above can be obtained at

https://devel.sharcnet.ca/repos/sharcware/src/fsignal

The reader is recommended to refer to the C version of signal handling for more details.

Technical Points Two things here need special attention. First there is a name convention universally accepted today in C/Fortran mixed languange programming. A C rouinte to be called from within Fortran programmes needs to have a trainling underscore in its name, as shown in the above example. If a C routine's name already contains an underscore or multiple underscores, adding one or two trailing underscore(s) is compiler dependent. GNU by default assumes two trailing underscores, while some other compilers such as Intel's assume a single underscore. Nvertheless most compilers have the option to specify whether to use a single trailing underscore.

Second in C function calls, arguments are passed by values, while in Fortran, arguments are passed by references. To call a C function from within a Fortran code, the arguments of the C function should be passed as pointers.

References

[1] Michael Metcalf and John K. Reid. *Fortran 90/95 Explained*. Oxford University Press, 2 edition, 1999.

[2] John Reid. Exception handling in Fortran. ACM Fortran Forum, 14, 9-15, 1995.