This assignment is an exercise in the use of signals and some of the process control system calls we have seen in class.

You are to write a program that asks a user to type in one character and press return. If the user types in ‘Q’ then your program should exit, and ‘return’ the character read in (here, ‘Q’). If the user types in ‘F’, then your program should perform a floating-point calculation that divides by 0. If the user types in ‘@’, then your program should send itself a SIGTERM. If the user types anything else, ignore it and ask them to type in another character.

Your program must contain signal handlers for SIGFPE and SIGALRM and not for SIGTERM i.e., the action for SIGTERM must be SIG_DFL. Each of your handlers should exit, ‘returning’ a status value of 3 for SIGFPE and 5 for SIGALRM. Each handler must print a message proving that it has caught the signal.

Your program should set an alarm for 2 seconds so that if a user has not typed in a character by that time, the alarm occurs, and your alarm handler runs. That means that each time a user types in a character you have to reset the alarm.

Now, once this is working, write another program which runs, as its child, the first program you’ve written. Show, in your program’s output, that you were in the parent process (include the process’ pid), then both the parent and the child (include the child’s pid) were running, and then that you have resumed the parent. In your output, be sure to demonstrate that the child process is, in fact, the child of the parent you are running (hint: see the man page for getppid()). Show that the parent receives

1. the exit status — 0 if the child exited normally, or the signal number that may have killed the child process (i.e., your child might exit for reasons other than the signals you handle: the parent should be able to find out if that happened) [hint: see the man page for wait()] and
2. the status information you arranged to ‘return’ via _exit().

For this assignment, you will submit one tar† file containing:

— the two programs (written in C or C++), one that is the parent, and the other that is the child,
— a README file that explains (1) how to compile the programs correctly [you might submit a makefile with your programs, then all you need here is to say how to use make], (2) how to run them, and (3) a sample of what running them looks like (i.e., you include output from having run the programs yourself).

You can develop the assignment on any UNIX-flavoured platform you like, but the version you submit must run correctly on the ITE UNIX server, which runs Solaris 8, and has gcc. Be aware, though, that programs that work on one flavour of UNIX may not work the same way, or at all, under another. It is your responsibility to ensure that what you submit works on the target platform.

About Floating-Point Exceptions

By default, the floating point hardware is ‘silent’ when it comes to floating point exceptions, like divide by 0.0. You will not only need to provide a signal handler for these exceptions, but also need to ‘arm’ the trap mechanism of the floating-point unit (FPU) hardware. A convenient way to do both in one step under Solaris (with Sun’s C compiler, cc, not the gnu compiler, gcc) is to use ieee_handler(). Read the fine manual page for that function.

About Signals and Handlers

UNIX provides several different “families” of functions for mapping signal handling action to signals (e.g.,

† “tar” (tape archive) is a standard UNIX facility for combining several files into one. See the course website (library section) and/or the man page for information on use of tar. ftp users take note: tar files are binary files.
SIG_IGN, SIG_DFL, your own handler). Generally, you should stick to using one “family” for your work. One such has the function `signal(pid, signum)`; another has the more elaborate function I mentioned in class, `sigaction()`. For sending signals to processes, the simplest function is probably `kill(pid, signum)` which sends signal `signum` to process `pid`. All of these functions are described in the manual (man) pages.

**Optional Bonus**

Normally, when a signal handler performs a return, control will return to the point in the process whence the handler was invoked. In some instances, the point in a program to which a signal handler returns should be different than the point from which the signal handler was invoked. For 10 bonus points, add a handler for SIGINT (which your process should receive when you type ctrl-C) that asks the user if they really want to quit and if they say no, then returns to your process at the point where it issues its first message to the user asking them to type a character and press return. If they say yes, the handler exits, ‘returning’ a value of 7.