ECM2433: The C family

C Workshop 5: forks, pipes and signals

In the lecture we looked at a simple program that responds to a signal. We ran the program in the background using a command like

```
myProgram &
```

The right-hand number that is displayed by Linux when you do this is the Process ID (PID) of the background process. The signal we used was a user-defined one called SIGUSR1, and we sent the signal to the program by issuing the following command on the Linux command line:

```
kill –USR1 <pid>
```

where <pid> is the PID.

Task 1: Write a program that traps the SIGUSR1 error. When the signal is received by your program it should just print out "SIGUSR1 received". Use the sleep function (you will need to #include stdlib.h) to make your program wait for the signal.

Task 2: There are many different signals; you can find their definitions on the GNU website, here:

https://www.gnu.org/software/libc/manual/html_node/Standard-Signals.html#Standard-Signals

One of these signals is SIGFPE, which is *automatically* raised when your program encounters a fatal arithmetic error.

Extend the program you wrote in task 1 to trap the SIGFPE error. To start with, write the main function as follows:

```
int main(void)
{
   int i;
   int x;

   /* your code from task 1 */

   for (i=-5; i<=5; i++)
   {
      x = 10/i;
      printf("%d: x = %d\n",i,x);
   }

   printf("Finished\n");
   return 0;
}
```

Run the program in the background. Raise the SIGUSR1 signal against it. What happens when you run this program?

The C runtime does not like integer divide-by-zero. Now amend the program so that it traps the SIGFPE signal. When the signal is raised, your program should print out a message saying "I have detected an arithmetic error and am giving up." and exit the program with an error.

Trapping this error in this way allows your program to do a bit of tidying up before exiting, rather than just crashing.

Task 3: On the ELE web page you will find the code for a program called forkPipe.c. This spawns a child process and send messages to it, which the child prints out. Copy this program and run it as a background process. You will get two warning messages when you compile it (in lines 53 and 63: "warning: initialization makes pointer from integer without a cast"); you can ignore them.

Make sure you understand what is happening by changing what the parent and child processes do.