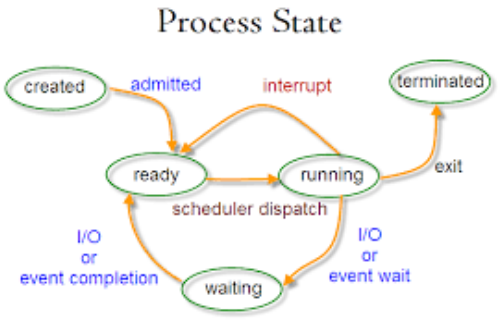
CS 201: Computer Systems

Assignment 3: "Family Feud" Shootout with Signals



Academic Integrity

You may NOT, under any circumstances, begin a programming assignment by looking for completed code on StackOverflow or Chegg or any such website, which you can claim as your own. Please check out the Student Code of Conduct at PCC.

The only way to learn to code is to do it yourself. The assignments will be built from examples during the lectures, so ask for clarification during class if something seems confusing. If you start with code from another source and just change the variable names or other content to make it look original, you will receive a zero on the assignment.

I may ask you to explain your assignment verbally. If you cannot satisfactorily explain what your code does, and answer questions about why you wrote it in a particular way, then you should also expect a zero.

Purpose

The purpose of this assignment is to be able to understand Processes and write signal handlers that can process signals between parent and child processes.

After completing this assignment you will be able to:

- Explain how signals and signal handlers work.
- Understand how you can block signals from your C code by using a series of functions for blocking signals.
- Understand how to write and call Signal Handlers.

Task

- We are going to program a shootout between child and parent processes which basically goes like this:
    - For 10 rounds, both child and parent will be sending each other signals. After sending a signal, each process will sleep for a random number of secs (from 1 to 3 seconds is about right).
    - When a process gets a signal, inside the signal handler it generates a random number (I used values between 1 and 50) as the amount of damage inflicted.
    - NOTE: Your signal handler should be very short. I set a global variable and let the main code handle the signal inside the loop (just check for damage not being zero). So your handler should be very short (only one or two lines).
    - To avoid getting hit again while processing a previous hit; block the signal first and restore it afterwards. See **sigblock.c** on how to block a signal. To restore the signal use at the end of processing:
      `sigprocmask(SIG_SETMASK,&prev_mask,&mask);`
    - Remember to set the global variable back to 0 after the processing is done.
    - NOTE: Only block the signal while processing a hit: check for a hit, then block signal, process the hit, then unblock signal.
    - To process a hit, keep track in each process how much damage has been done (this can be a local variable outside of the loop). If a process exceeds a maximum damage (I used 200) then they have 'died'. Have the child either quit using _exit or just break out of the loop. The parent should break out and then wait for the child to end so it can be 'reaped'.
    - If a process manages to get through all rounds, then it survives. Note that either process may both live or both die as well as one living and one dying.
    - HINT: Look at the **sigchild.c** example as a starting point. Note you'll need to put in a loop so you can get several rounds in.
    - Using the rand() function in stdlib.h library (include this), you will need to generate random numbers between 1 and 3 for the amount to sleep before shooting again.
    - The rand() function:
        - The rand() function will return a value between 0 and the largest possible int value. This is way too big a value so often you need to use a modulo operator to cut down the size to something reasonable. For our purposes we need to get a number between 1 and 3. To do this, use the following line of code:
          `int sleep = (rand() % 3) + 1;`
    - The srand() function:
        - Before the code can call the rand() function, the srand() function needs to be called at least once in your code for each process. This will start the generator with a different start number. If this is not done, the series of random numbers will be the same every time the program is run for both

processes which are not very random. So make sure you call srand with a different number for each process before calling the rand function.

- Note that if you use a 'hard coded' number for srand it will start with the same sequence of numbers. So in order to get a different set of numbers each time it is run, the standard way is to use the time() function. This requires including the time.h library header. The time() function, when given a 0 (or NULL), passes back the current number of seconds since 1970 as a signed long. This needs to be typecasted to an unsigned for the srand function.
- To get each process to have a different set of numbers, you can try multiplying the time() value by a small constant. This is not perfect but seems to give a different set of numbers between parent and child. I used 3 for the child and 7 for the parent. So after the fork before going into the event loop:
  `for the parent: srand((unsigned)time()*7);`
  `for the child: srand((unsigned)time()*3);`
- Make sure you only call srand() once. If you reset the seed each time there is a tendency of getting very similar random numbers so it's best to just call it once and then call only rand() when you need to put the process to sleep.

Criteria for Success

- For information on how to do this assignment, see week 7 and week 8 lecture pages on Processes and Signal Handlers in the D2L shell.
- Name your source code, **`shootout.c`**. You can start from scratch or use **`sigchild.c`** as a starting point.
- Test your program using the given sample runs or your own data multiple times.
- Upload the C source code file per the instructions in the D2L Assignment folder for assignment 3 by the due date. ***Some instructors require you to turn in your assignment on D2L while some others like it turned into the Linux server. Be sure to read the instructions in the assignment folder.***
- Do your own work. Consult the syllabus for more information about academic integrity.

Sample Runs

```
====================
OUTPUT ONE
====================
child 12692 got hit with +45. damage is now 45
parent 12691 got hit with +36. damage is now 36
child 12692 got hit with +43. damage is now 88
parent 12691 got hit with +40. damage is now 76
parent 12691 got hit with +14. damage is now 90
```

```
child 12692 got hit with +25. damage is now 113
parent 12691 got hit with +43. damage is now 133
child 12692 got hit with +25. damage is now 138
parent 12691 got hit with +24. damage is now 157
child 12692 got hit with +12. damage is now 150
parent 12691 got hit with +39. damage is now 196
child 12692 got hit with +45. damage is now 195
parent 12691 got hit with +44. damage is now 240
parent has died!
child 12692 got hit with +31. damage is now 226
child has died!
```

```
====================
OUTPUT TWO
====================
child 12720 got hit with +3. damage is now 3
parent 12719 got hit with +25. damage is now 25
child 12720 got hit with +29. damage is now 32
parent 12719 got hit with +26. damage is now 51
child 12720 got hit with +41. damage is now 73
parent 12719 got hit with +30. damage is now 81
child 12720 got hit with +45. damage is now 118
parent 12719 got hit with +18. damage is now 99
child 12720 got hit with +47. damage is now 165
parent 12719 got hit with +4. damage is now 103
child 12720 got hit with +32. damage is now 197
parent 12719 got hit with +48. damage is now 151
child 12720 got hit with +20. damage is now 217
child has died!
parent 12719 got hit with +1. damage is now 152
parent has survived!
```

```
======================
OUTPUT THREE
======================
parent 12855 got hit with +44. damage is now 44
child 12856 got hit with +42. damage is now 42
child 12856 got hit with +46. damage is now 88
parent 12855 got hit with +29. damage is now 73
child 12856 got hit with +36. damage is now 124
child 12856 got hit with +9. damage is now 133
parent 12855 got hit with +30. damage is now 103
parent 12855 got hit with +36. damage is now 139
```

```
child 12856 got hit with +24. damage is now 157
parent 12855 got hit with +41. damage is now 180
child 12856 got hit with +9. damage is now 166
parent 12855 got hit with +18. damage is now 198
child 12856 got hit with +18. damage is now 184
parent 12855 got hit with +29. damage is now 227
child 12856 got hit with +34. damage is now 218
parent has died!
child has died!
```

Grading Criteria

- Compiles in gcc without errors.
- Program runs without run-time errors.
- Program uses a signal handler and sets a global flag variable.
  NOTE: global flag must be declared volatile **`sig_atomic_t`** type to prevent unsafe asynchronous behavior.
- Program sends signals to the other process using the **`kill`** function. Calculations should done **in the process that receives the signal (for both parent and child) not in the process that sends it**.
- The program uses the **`rand`** function to variate the sleep times.
- The **`srand`** function is set with a different number each time the program is run.
- The behavior should be pretty close to the output given. However, it won't be exact because of the randomness in the program. Also, you have some leeway in what you would like to print out as well, such as making a 'star wars' or 'star trek' theme, etc. This is encouraged if you have the time :)