**CS 4760** | **Operating Systems**
Assignment 2 | Due Date: September 26, 2018

---

# Concurrent UNIX Processes and shared memory

The goal of this homework is to become familiar with using shared memory and creating multiple processes.

**Problem:** In this project you will have two executable files. The first executable file, Master, will be in charge of launching a specific number of child processes using a `fork` followed by an `exec`. Master should then keep track of how many children have finished executing and terminate itself only when all of its children have finished.

Master should take in several command line options. First, -h, to describe how it should be run. Then a -n x to indicate the maximum total of child processes to fork off. Another option, -s, will indicate how many children should be allowed to exist in the system at the same time.

So for example, if called with -n 10 -s 5 then Master would want to fork/exec off 5 child processes but then not create any more until some of them started to terminate. Once one had terminated, it would create another, continuing this until it had created a total of 10. Then at that point it would wait until all of them had terminated.

As in this project we want to get familiar with shared memory, Master will also be responsible for creating shared memory to store two integers and then initializing those integers to zero. This shared memory should be accessible by the children. This shared memory will act as a ”clock” in that one integer represents seconds, the other one represents milliseconds. Note that this is our ”simulated” clock and will not have any correlation to real time.

There are several termination criteria. First, if all the children have finished, master should free up shared memory and terminate. In addition, I expect your program to terminate after 2 real life seconds. This can be done using a timed signal, at which point it should kill off all currently running child processes and terminate. It should also catch the ctrl-c signal, free up shared memory and then terminate all children. No matter how it terminated, master should also output the value of the shared clock.

**The children** The task in the child executable (let us call it worker) is fairly straightforward. They will be given a command line argument when they were execed off, this should be the value of n (in our example it was 10). Then they should try and increment the value of the clock in shared memory by that value multiplied by a million. After each access they should check to see if the result ”overflows” the counter of milliseconds. If so, they should increment the seconds appropriately and subtract the corresponding amount of milliseconds. Once they are done doing this, output a message to the screen of how much they have incremented and then terminate.

I suggest you implement these requirements in the following order:

1. Get a makefile that compiles two source files
2. Have master allocate shared memory, use it, then deallocate it. Make sure to check all possible error returns.
3. Get Master to fork off and exec one child and have that child attach to shared memory and do something with the clock. Master should wait for it to terminate. Then when the child terminates Master should output the value of the clock.
4. Put in the signal handling to terminate after 2 seconds. A good idea to test this is to simply have the child go into an infinite loop so Master will not ever terminate normally. Once this is working have it catch Ctrl-c and free up the shared memory, send a kill signal to the child and then terminate itself.
5. Lastly start setting up code to fork off child processes until the specific limits.

Make sure you never have more than 20 processes in the system at any time, even if the program is invoked with $n$ being more than 20.

**Hints**

You will need to set up shared memory in this project to allow the processes to communicate with each other. Please check the man pages for `shmget`, `shmctl`, `shmat`, and `shmdt` to work with shared memory.

You will also need to set up signal processing and to do that, you will check on the functions for `signal` and `abort`. If you abort a process, make sure that the parent cleans up any allocated shared memory before dying.

In case you face problems, please use the shell command `ipcs` to find out any shared memory allocated to you and free it by using `ipcrm`.

## What to handin

I will want you to hand in an electronic copy of all the sources, `README`, Makefile(s), and results. Create your programs in a directory called *username*.1 where *username* is your login name on hoare. Once you are done with everything, *remove the executables and object files*, and issue the following commands:

```
chmod 700 username.2

cp -p -r username.2 /home/hauschild/cs4760/assignment2
```

Do not forget `Makefile` (with suffix or pattern rules), `RCS` (or some other version control like Git), and `README` for the assignment. If you do not use version control, you will lose 10 points. I want to see the log of how the file was modified. Omission of a `Makefile` will result in a loss of another 10 points, while `README` will cost you 5 points. Make sure that there is no shared memory left after your process finishes (normal or interrupted termination). Also, use relative path to execute the child.