CS 4760 **Operating Systems**
Assignment # 3 Due Date: March 16, 2020

Semaphores and Message Passing

The goal of this homework is to become familiar with semaphores in the Unix operating system. In addition, you will implement a concept, provide it as a statically linked library, and use it to solve the following problem.

Problem: Write a program to compute the summation of n integers in each of the following ways and assess their performance. Assume that n is a power of 2.

1. Partition the n integers into $n/2$ pairs. Use $n/2$ processes to add together each pair of integers resulting into $n/2$ integers. Repeat the method on the $n/2$ integers and continue until the final result is obtained. (This is a binary tree algorithm.)
2. Divide the n integers into $n/\log n$ groups of $\log n$ numbers each. Use $n/\log n$ processes, each adding the numbers in one group sequentially. Then add the $n/\log n$ results using method 1.

Structure the integers into a file, one integer per line.

The main program will set up shared memory to read integers, set up a signal handler, and then, read all integers into shared memory from specified file. Since you do not know the number of integers to read, you may have to run two passes over the data file to find the number of integers and to actually read in the integers. Then, the main program will fork off child processes who will create more children as needed. The children will test the index assigned to them, perform the computation, and write the result in place of the first integer. Further, the child process will also update a log file, named `adder_log` to say what has been done. You will use semaphores to protect the critical resources – the log file.

Make sure you never have more than 20 processes in the system at any time. Add the pid of the child to the file as comment to the file. The preferred output format is:

```
PID   Index    Size
```

The child process will be `exec`ed by the command

```
bin_adder xx yy
```

where `xx` is the index number starting the list of integers to be added in shared memory and `yy` is the number of integers to be added.

If a process starts to execute code to enter the critical section, it must print a message to that effect on `stderr`. It will be a good idea to include the time when that happens. Also, indicate the time on `stderr` when the process actually enters and exits the critical section. Within the critical section, wait for 1 second before you write into the file, and then, wait for another 1 second before leaving the critical section. For each child process, tweak the code so that the process requests and enters the critical section at most five times.

The code for each child process should use the following template:

```
for ( i = 0; i < 5; i++ )
{
    sleep for random amount of time (between 0 and 3 seconds);
    wait for semaphore;
    /* Critical section */
    signal semaphore;
}
```

Implementation

You will be required to create separate `bin_adder` processes from your main process. That is, the main process will just spawn the child processes and wait for them to finish. The *main* process also sets a timer at the start of computation to 100 seconds. If computation has not finished by this time, the *main* process kills all the spawned processes and then exits. Make sure that you print appropriate message(s).

In addition, the main process should print a message when an interrupt signal (`^C`) is received. All the children should be killed as a result. All other signals are ignored. Make sure that the processes handle multiple interrupts correctly. As a precaution, add this feature only after your program is well debugged.

The code for main and child processes should be compiled separately and the executables be called `master` and `bin_adder`. Make sure that you do not use absolute path for the child processes.

Other points to remember: You are required to use `fork`, `exec` (or one of its variants), `wait`, and `exit` to manage multiple processes. Use `shmctl` suite of calls for shared memory allocation. Also make sure that you do not have more than twenty processes in the system at any time. You can do this by keeping a counter in `master` that gets incremented by `fork` and decremented by `wait`.

What to handin

Handin an electronic copy of all the sources, `README`, Makefile(s), and results. Create your programs in a directory called *username*.3 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:

```
% cd
% chmod 755 ~
% ~sanjiv/bin/handin cs4760 3
% chmod 700 ~
```

Create your own input data file, with at least 64 integers, one on each line, using a random number generator. Let the integers range between [0,256). Comment on your comparison of the two methods'performance in `README`.

Do not forget `Makefile` (with suffix rules), version control, and `README` for the assignment. If you do not use version control, you will lose 10 points. I want to see a log of how the project got modified. Omission of a `Makefile` (with suffix rules) will result in a loss of another 10 points, while `README` will cost you 5 points. Make sure that there is no IPC structures left after your process terminates (normal or interrupted termination). Also, use relative path to execute the child. Run the problem for prescribed time and kill everything at that point if it is not completed. Also print messages when a process waits for and acquires a semaphore using a logfile.