CS 4760 **Operating Systems**

Assignment # 3 Due Date: March 19, 2019

Semaphores and Operating System Simulator

Problem: As Kim knew from her English courses, palindromes are phrases that read the same way right-to-left as they do left-to-right, when disregarding capitalization. Being the nerd she is, she likened that to her hobby of looking for palindromes on her vehicle odometer: 245542 (her old car), or 002200 (her new car).

Now, in her first job after completing her undergraduate program in Computer Science, Kim is once again working with Palindromes. As part of her work, she is exploring alternative security encoding algorithms based on the premise that the encoding strings are *not* palindromes. Her first task is to examine a large set of strings to identify those that are palindromes, so that they may be deleted from the list of potential encoding strings.

The strings consist of letters, $(a, \ldots, z, A, \ldots Z)$, and numbers $(0, \ldots, 9)$. Kim has a one-dimensional array of pointers, `mylist[]`, in which each element points to the start of a string (You can use `string` class of C++ if you so desire). Each string terminates with the `NULL` character. Kim's program is to examine each string and print out all string numbers (subscripts of `mylist[]`) that correspond to palindromes in one file while the non-palindromes in another file.

Your job is obviously to write the code for Kim. The main program will read the list of palindromes from a file (one string per line) into shared memory and fork off processes. The children will test the index assigned to them and write the string into appropriate file, named `palin.out` and `nopalin.out`. You will have to use the semaphores to protect the critical resources – the two files.

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. Add the pid of the child to the file as comment in the log file. The preferred output format for log file is:

```
PID   Index    String
```

where `Index` is the logical number for consumer process, assigned internally by your code, and varies between 0 and $n - 1$.

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

```
palin xx
```

where `xx` is the index number of the string to be tested in shared memory. You can supply other parameters in `exec` as needed.

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 2 seconds before you write into the file, and then, wait for another 2 seconds 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);
    execute code to enter critical section;
    /* Critical section */
    execute code to exit from critical section;
}
```

Implementation

You will be required to create separate `consumer` 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

25 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 and child processes should be compiled separately and the executables be called `master` and `palin`.

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:

```
chmod 700 username.3

cp -p -r username.3 /home/hauschild/cs4760/assignment3
```

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.