CS 4760 **Operating Systems**
Assignment 2 Due Date: October 02, 2020

Concurrent UNIX Processes and shared memory

Purpose

The goal of this homework is to become familiar with concurrent processing in Linux using shared memory.

Task

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 $(0, \ldots, 9)$. Ignore any punctuations and spaces. 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, but I think it will be easier to use C strings for this project). Each string terminates with the `NULL` character. Kim's program is to examine each string and print out all strings that are palindromes in one file while the non-palindromes are written to 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. Each child will test the string at the index assigned to it and write the string into appropriate file, named `palin.out` and `nopalin.out`. In addition, the processes will write into a log file (described below). You will have to use the code for multiple process IPC problem (Solution 4 in notes) 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 $n > 20$. 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 the child, 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, such as the logical number of the child.

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 [0-2] seconds before you write into the file.

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

```
    determine if the string is a palindrome;
    execute code to enter critical section;
    sleep for random amount of time (between 0 and 2 seconds);
    /* Critical section */
    execute code to exit from critical section;
```

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

In addition, `master` 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`.

Invoking the solution

`master` should take in several command line options as follows:

```
master -h
master [-n x] [-s x] [-t time] infile
```

`-h`	Describe how the project should be run and then, terminate.
`-n x`	Indicate the maximum total of child processes `master` will ever create. (Default 4)
`-s x`	Indicate the number of children allowed to exist in the system at the same time. (Default 2)
`-t time`	The time in seconds after which the process will terminate, even if it has not finished. (Default 100)
`infile`	Input file containing strings to be tested.

Note that all of these options should have some sensible default values, which should be described if `-h` is specified. For example, if the project is called with: `master -n 5 -s 2 palindromes` then `master` will fork/exec 2 child processes but then not create any more until one of them terminated. Once a child terminates, it'll create another, continuing this until it has created a total of 5 children. At that point it will wait until all of them have terminated. When done, it would output appropriate information to the logfile, called `output.log`.

Termination Criteria: There are several termination criteria. First, if all the children have finished, the parent process should deallocate shared memory and terminate.

In addition, I expect your program to terminate after the specified amount of time (`-t` option) of real clock. This can be done using a timeout signal, at which point it should kill 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 terminates, `master` should also output the value of the shared clock to the output file. For an example of a periodic timer interrupt, you can look at p318 in the text, in the program `periodicasterik.c`.

Suggested implementation steps

I suggest you implement these requirements in the following order:

1. Get a `Makefile` that compiles the two source files.

2. Have your main executable read in the command line arguments and output the values to your screen (and ensure `-h` displays useful results).
3. Create and test the function to determine if a provided string is a palindrome.
4. Have `master` allocate shared memory, use it, then deallocate it. Make sure to check all possible error returns.
5. Get `master` to fork and exec one child and have that child attach to shared memory and read the clock. Have the child output any information it was passed from master and then the value of the clock to test for correct launch. `master` should wait for it to terminate and then output the value of the clock at that time.
6. Put in the signal handling to terminate after specified number of 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.
7. Set up the code to fork multiple child processes until the specific limits.
8. Make each child process test the given string for being a palindrome, and write into appropriate file.
9. Ensure that the critical section problem is solved.

If you do it in this order, incrementally, you help make sure that the basic fundamentals are working before getting to the point of launching many processes.

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. This is a hard limit.

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`.

Please make any error messages meaningful. The format for error messages should be:

```
master: Error: Detailed error message
```

where `master` is actually the name of the executable (`argv[0]`) that you are trying to execute. These error messages should be sent to `stderr` using `perror`.

Criteria for success

You are free to use your own data structures and algorithms. Your code should properly test all the supplied strings. They should be output appropriately to the two specified files. *You must clean up after yourself.* That is, after the program terminates, whether normally or by force, there should be no shared memory left that is allocated to you.

Grading

1. *Overall submission: 25 pts.* Program compiles and upon reading, seems to be able to solve the assigned problem in the specified manner (shared memory/fork/exec).
2. *README: 5 pts.* Must address any special things you did, or if you missed anything.

3. *Makefile: 5pts.* Must use suffix rules or pattern rules. You'll receive only 2 points for `Makefile` without those rules.
4. *Command line parsing: 10 pts.* Program is able to parse the command line appropriately, assigning defaults as needed; issues help if needed.
5. *Use of perror: 5pts.* Program outputs appropriate error messages, making use of perror(3). Errors follw the specified format.
6. *Code readability: 10 pts.* The code must be readable, with appropriate comments. Author and date should be identified.
7. *Proper fork/exec/wait: 10 pts.* Code appropriately performs the fork/exec/wait functions. There are no zombie processes. The number of processes is limited as specified in command line options.
8. *Shared memory: 10 pts.* Shared memory is properly allocated/deallocated and used to transmit data from `master` to `palin`.
9. *Signals: 10 pts.* Code reacts to signals as specified. When the parent is terminated, all children are terminated as well, and shared memory deallocated.
10. *Conformance to specifications: 10pts.* Code properly creates the two files: `palin` and `nopalin`. The log file is appropriate, and appropriate messages are displayed to screen.

Submission

Handin an electronic copy of all the sources, `README`, `Makefile`(s), and results. Create your programs in a directory called *username*.2 where *username* is your login name on hoare. Once you are done with everything, *remove the executables and object files as well as log files*, and issue the following commands:

```
% cd
% chmod 755 ~
% ~sanjiv/bin/handin cs4760 2
% chmod 700 ~
```

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.

You do not have to hand in a hard copy of the project. Assignment is due by 11:59pm on the due date.