EECS 338 Homework 3: Multiprocess Programming

General requirements:

- Due on the posted due date.
- Upload a single, compressed file (e.g. zip) to Canvas that contains all required files.
- Include either a single makefile that compile all programs or a separate makefile for each.
- Include a typed document with a description of any techniques that you used to control the output order.
- All work should be your own, as explained in the Academic Integrity policy from the syllabus.

Instructions: The purpose of this assignment is to write more complex programs using fork() and concurrency.

1. Write a custom shell program as described in the textbook: Programming Project 1, Part 1 (page 157). Do not do Part 2 (history feature) unless you wish to do so for fun. You are not required to detect "&" for background processes unless you wish to do so for fun. The minimum requirements are:

   - Use the basic design provided in simple-shell.c that was provided in class and appears in Fig. 3.36.
   - Parse the user input appropriately. You may use the approach provided in execvp_demo.c that was provided in class, or you may use a different approach that is more robust.
   - Fork a new process for every command (except "exit") in order to provide separate process control.
   - Use wait() for the parent process.

   Below is an example of how your output might look:

   ```
   chris> echo This is my own shell!
   This is my own shell!
   chris> ls shell.c -all
   -rwxr-xr-x 1 chrisf None 1146 Sep 14 22:25 shell.c
   chris> ./sleep
   Sleeping
   chris> exit
   ```

2. Write a program that computes the run times for parallel computations similar to the examples shown in class (core_test_fork.c and core_test_fork4.c) with the following difference. Allow the user to enter the number of processes (from 1 to 4). For the computation, you may repeat the simple summation in the examples, or you may use another type of computation of your own choosing, provided that the computations are equally divided between the processes. Run the program for at least 1 - 4 processes. Type a report showing the run times and discuss your findings. Are they as you expected or not? Below is an example using parallel summations like the sample programs:

   ```
   $ ./compute
   How many processes? 1
   Process 1 total time was 5.426827 seconds. sum = 1000000000.
   $ ./compute
   How many processes? 2
   Process 2 total time was 3.372174 seconds. sum = 500000000.
   Process 1 total time was 3.400580 seconds. sum = 500000000.
   $ ./compute
   How many processes? 3
   Process 3 total time was 2.750818 seconds. sum = 333333333.
   Process 2 total time was 2.725256 seconds. sum = 333333333.
   Process 1 total time was 2.708162 seconds. sum = 333333333.
   $ ./compute
   ```

```
How many processes? 4
Process 4 total time was 2.270596 seconds. sum = 250000000.
Process 3 total time was 2.291373 seconds. sum = 250000000.
Process 2 total time was 2.303331 seconds. sum = 250000000.
Process 1 total time was 2.241928 seconds. sum = 250000000.
```

Special notes: You may control the forking with only "if" statements, or you may challenge yourself to use a loop. The following are suggestions only and are not required: You may need to use a local computer to see a speed increase, as the server may not show any differences in speed. You might use "lscpu" or some other method to determine the number of cores on the computer. In the example above, wait() was used to control the output order, and each process printed out an index number, but this is not required.

3. Create a single program that creates shared memory for a string, forks a child process that writes the string to the shared memory, and reads/prints the value created by the child. This is similar to the examples from class (shm_producer.c and shm_consumer.c). However, there is only one program, and you will create and unlink the memory in the parent process. The child can automatically have the same pointer. Below is an example of how your output might look:

```
Hello from your child!
```

Special notes: If you use the EECS server, we recommend that you change the name for the memory from "OS" to something unique to avoid conflicts on the server. It is not necessary to put each word in a separate string as the textbook example does (you can use a string with spaces and multiple words). You may see a warning about using the void pointer. You may ignore it or change the type to char''. To compile on the EECS server, you will need to link the Realtime Extensions library, as explained in class.

For fun (not for credit): Rewrite Problem 2 above to use shared memory to store the sum from each child, and have the parent compute and display the total sum. Hints: make your pointer type "long long int" and store each child's sum in ptr[0], ptr[1], etc. You may need to use waitpid() to ensure that all children finish before the parent tries to use the values.

Rubric:

Item	Points
Problem 1: Basics - compiles, general programming expectations	5
Problem 1: Overall program design, including while loop	5
Problem 1: Handling exit case	5
Problem 1: Handling parent vs. child	10
Problem 1: Calling fork, execvp, and wait	10
Problem 2: Basics - compiles, general programming expectations	5
Problem 2: Overall program design	5
Problem 2: forking logic	10
Problem 2: Dividing computations evenly	5
Problem 2: Typed report any reasonable output and discussion	5
Problem 3: Basics - compiles, general programming expectations	5
Problem 3: Setting up shared memory	5
Problem 3: Calling fork, wait, and shm_unlink	10
Problem 3: Handling parent vs. child	10
Problem 3: Writing string child,and reading string parent	5
Total	100