Homework 3

WORKING WITH SHARED MEMORY AND PIPES FOR INTER PROCESS COMMUNICATION

The objective of this assignment is to get comfortable with Inter Process Communication (IPC) using Shared Memory and Pipes. These approaches are among two of the most dominant mechanisms for doing IPC. Familiarity with shared memory will also help you with some of the advanced concepts that we will cover in process synchronization.

DUE DATE: Wednesday, February 26th @ 8:00 pm MT

Generative AI Use and Consequences
Use of AI tools such as ChatGPT, Claude, Github Co-Pilot, and/or their ilk to write or "improve" your code or written work at *any* stage is prohibited; this includes the ideation phase. It is your responsibility to ensure that you don't have the GitHub Co-Pilot extension installed in your IDE; assignment solutions generated by Co-Pilot aren't written by you. Turning in code or an essay written by generative AI tools will be treated as turning in work created by someone else, namely an act of plagiarism and/or cheating.

Ultimately, you will get out of the class what you put in. Simply copying and pasting code from generative AI tools is neither ethical nor does it contribute to your learning experience. There are multiple reasons why these generative AI tools are detrimental to your learning experience:

1. They rob you of the ability to think and learn the concepts for yourself. Solving problems is an essential step to gaining a solid understanding of the material.
2. You will struggle with the in-classroom quizzes and exams where you will not have access to these tools.
3. While we acknowledge that these tools are likely to become an important part of a software engineer's workflow in the future, you are much more likely to use these tools in an effective manner if you already have expertise in the relevant technical topics. Developing such expertise requires putting in the effort to learn these topics without the assistance of these tools.
4. These tools are prone to generating imperfect or even incorrect solutions, so trusting them blindly can lead to bad consequences.

Some helpful Infospaces videos for this assignment:

(1) Pipes in C: https://infospaces.cs.colostate.edu/watch.php?id=279.

(2) Shared Memory in C: https://infospaces.cs.colostate.edu/watch.php?id=280

1 Description of Task

This assignment builds on HW2. Specifically, we will be using IPC for communications between the Coordinator and Checker programs.

The **Coordinator** behaves similarly to the previous assignment, but has the following new capabilities:

1. Creation of unique shared memory segments for each Checker instance to store results.
2. Creation of a pipe for each Checker instance that provides it with the ID of the shared memory segment created in step (1). The file descriptor (FD) of the pipe is passed as an additional argument to the Checker.
3. Checker processes run concurrently rather than sequentially. This means that the Coordinator will launch all the child processes and **then** start waiting for results.

As in the previous assignment, each instance of the Checker will receive different arguments. To facilitate this, the Coordinator will take a total of five command line arguments and selectively pass them on to the Checker. The first argument is the divisor, followed by the dividends. For instance,

```
> coordinator 3 8 15 21 45
```

Would create 4 child processes that would check 8/3, 15/3, 21/3, and 45/3, respectively, in parallel.

The **Checker** has changed as well:

1. An additional command line argument gives the FD of the pipe to read from.
2. Using the pipe FD, the Checker determines the segment ID of the shared memory to store its result.
3. Rather than returning the result of the check, the result is stored in the shared memory segment.

The two arguments that the `Checker` needs to perform its mathematical operation as well as the pipe FD will be supplied to it by the `Coordinator;` the `Coordinator` is supplied these aforementioned arguments from the command line.

All print statements must indicate the program that is responsible for generating them. To do this, please prefix your print statements with the program name i.e. `Coordinator` or `Checker`. The example section below depicts these sample outputs.

2 Requirements of Task

The following requirements have been updated from HW2.

1. The Checker must accept **three** arguments, and the Coordinator must accept five command line arguments.
2. The Coordinator creates a pipe using the **`pipe()`** command for each child process. The read end of the pipe will be passed to the Checker, and the write end of the pipe will be used by the Coordinator to provide the shared memory segment ID.
3. The Coordinator should spawn 4 processes using the **`fork()`** command and print their process IDs as they are created.
4. Child-specific processing immediately following the **`fork()`** command loads the Checker program into the newly created process using the **`exec()`** command. This ensures that the forked process is no longer a copy of the Checker. This **`exec()`** command should also pass 3 arguments to the Checker program: the divisor, dividend, and the FD of the read end of the pipe created in (2).
5. The Coordinator sets up the shared memory using **`shmget()`** and writes its ID to the pipe.
6. The Checker starts executing, prints out its process ID, and retrieves the shared memory segment ID from the pipe.
7. The `Checker` determines whether or not argTwo is divisible by argOne and prints this information.
8. If divisible, `Checker` should write **1** (true) to the shared memory segment, or **0** (false) otherwise.
9. Parent-specific processing in the `Coordinator` should ensure that the `Coordinator` will **`wait()`** for each instance of the child-specific processing to complete. This is done **AFTER** all the processes have been started. The results retrieved from shared memory should be printed and match up with what was printed in (7).
10. Both the Coordinator and Checker should clean up: FDs should be closed and shared memory marked to be destroyed (use the **`shmctl()`** command).

Figure 1 below depicts the assignment scenario. The first step is for the Coordinator will complete multiple cycles of {fork, exec, and wait} for each Checker process. Multiple Checker process will be active. The second step is the controller creates Shared Memory and Pipe. The third step is the Pipe FD is passed via command line to the Checker. Fourth, the segment ID is passed via the Pipe to the Checker. Fifth, is the results are computed by the Checker are written to Shared memory and read by the Coordinator.

1. The `Coordinator` will complete multiple cycles of **{`fork`, `exec`,** and **`wait`}** for **each** `Checker` process. **Multiple** `Checker` processes will be active.
2. The `Controller` creates the **Shared Memory** and **Pipe**
3. The Pipe FD is passed via command line to the `Checker`
4. The segment ID is passed via the Pipe to the `Checker`
5. Results computed by the `Checker` are written to Shared memory and read by the `Coordinator`

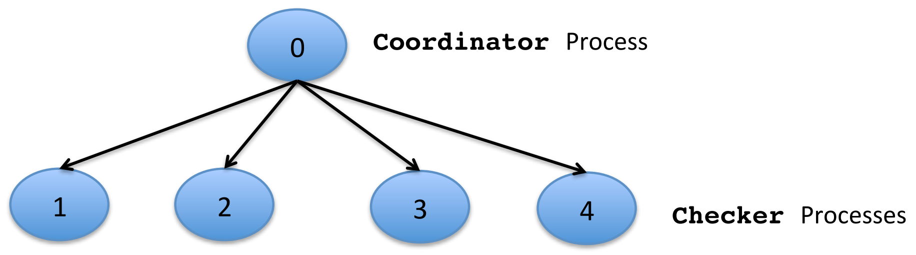

Figure 1: Visual representation of the assignment goals

3 Example Output:

```
> ./coordinator 3 3 20 49 102
Coordinator: forked process with ID 25496.
Coordinator: wrote shm ID 4063308 to pipe (4 bytes)
Coordinator: forked process with ID 25497.
Coordinator: wrote shm ID 4096077 to pipe (4 bytes)
Coordinator: forked process with ID 25498.
Coordinator: wrote shm ID 4128846 to pipe (4 bytes)
Coordinator: forked process with ID 25499.
Coordinator: wrote shm ID 4161615 to pipe (4 bytes)
Coordinator: waiting on child process ID 25496...
Checker process [25498]: starting.
Checker process [25498]: read 4 bytes containing shm ID 4128846
Checker process [25498]: 49 *IS NOT* divisible by 3.
Checker process [25498]: wrote result (0) to shared memory.
Checker process [25497]: starting.
Checker process [25497]: read 4 bytes containing shm ID 4096077
Checker process [25497]: 20 *IS NOT* divisible by 3.
Checker process [25497]: wrote result (0) to shared memory.
Checker process [25499]: starting.
Checker process [25499]: read 4 bytes containing shm ID 4161615
Checker process [25499]: 102 *IS* divisible by 3.
Checker process [25499]: wrote result (1) to shared memory.
Checker process [25496]: starting.
Checker process [25496]: read 4 bytes containing shm ID 4063308
Checker process [25496]: 3 *IS* divisible by 3.
Checker process [25496]: wrote result (1) to shared memory.
Coordinator: result 1 read from shared memory: 3 is divisible by 3.
Coordinator: waiting on child process ID 25497...
```

```
Coordinator: result 0 read from shared memory: 20 is not divisible by 3.
Coordinator: waiting on child process ID 25498...
Coordinator: result 0 read from shared memory: 49 is not divisible by 3.
Coordinator: waiting on child process ID 25499...
Coordinator: result 1 read from shared memory: 102 is divisible by 3.
Coordinator: exiting.

> ./coordinator 7 32 49 846 22344
Coordinator: forked process with ID 25504.
Coordinator: wrote shm ID 4194380 to pipe (4 bytes)
Coordinator: forked process with ID 25505.
Coordinator: wrote shm ID 4227149 to pipe (4 bytes)
Coordinator: forked process with ID 25506.
Coordinator: wrote shm ID 4259918 to pipe (4 bytes)
Coordinator: forked process with ID 25507.
Coordinator: wrote shm ID 4292687 to pipe (4 bytes)
Coordinator: waiting on child process ID 25504...
Checker process [25504]: starting.
Checker process [25504]: read 4 bytes containing shm ID 4194380
Checker process [25504]: 32 *IS NOT* divisible by 7.
Checker process [25504]: wrote result (0) to shared memory.
Coordinator: result 0 read from shared memory: 32 is not divisible by 7.
Coordinator: waiting on child process ID 25505...
Checker process [25506]: starting.
Checker process [25506]: read 4 bytes containing shm ID 4259918
Checker process [25506]: 846 *IS NOT* divisible by 7.
Checker process [25506]: wrote result (0) to shared memory.
Checker process [25507]: starting.
Checker process [25505]: starting.
Checker process [25507]: read 4 bytes containing shm ID 4292687
Checker process [25505]: read 4 bytes containing shm ID 4227149
Checker process [25507]: 22344 *IS* divisible by 7.
Checker process [25507]: wrote result (1) to shared memory.
Checker process [25505]: 49 *IS* divisible by 7.
Checker process [25505]: wrote result (1) to shared memory.
Coordinator: result 1 read from shared memory: 49 is divisible by 7.
Coordinator: waiting on child process ID 25506...
Coordinator: result 0 read from shared memory: 846 is not divisible by 7.
Coordinator: waiting on child process ID 25507...
Coordinator: result 1 read from shared memory: 22344 is divisible by 7.
Coordinator: exiting.
```

4 What to Submit

Assignments should be submitted through Canvas. E-mailing the codes to the Professor, GTA, or the class accounts will result in an automatic 1 point deduction.

Use the CS370 *Canvas* to submit a single .zip file that contains:

- All .c and .h files related to the assignment (please document your code),
- A Makefile that performs both a *make clean* as well as a *make all,*
- A README.txt file containing a description of each file and any information you feel the grader needs to grade your program.

Filename Convention: Your coordinator and checker must be named coordinator.c and checker.c respectively; you can name additional .c and .h files anything you want. The archive file should be named as <LastName>-<FirstName>-HW3.zip. E.g. if you are Cameron Doe and submitting for HW3, then the zip file should be named Doe-Cameron-HW3.zip.

5 Grading

This assignment would contribute a maximum of 5 points towards your final grade. The grading will also be done on a 5-point scale. The points are broken up as follows:

0.5 point each for each of the tasks i.e. Task 1-10 (**5 points**)

You are required to work alone on this assignment.

6 Late Policy

All assignments are due at 8:00 PM on the due date. There is a late penalty of 10% per-day for up to a maximum of 2 days.