CMSC 405: Assignment 1: IPC Mechanisms

Due date: Monday Feb 23rd (midnight) 2026.

Question 1: Shared Memory (15 pts):

Write a simple program (example code available on Canvas; can be used as starting point) where *processes synchronize via polling*. Consider THREE processes: A, B and C. Process A is the master process and will print out the strings from *shared memory* written by two other writing processes (B first and then C second). Process A needs to 'wait' by polling until B and C finish writing their strings to memory. Each of processes A, B and C should be in different code files. Create THREE different shared memory locations: (i) one for storing the pid, (ii) for storing the process identifier (A, B or C) and (iii) for storing the string. Here is the sequence of events that needs to be implemented:

1. Process A writes "<pid>" to the first shared memory location and "A" to the second location. It also writes the string "I am Process A" into the string shared memory location. It then prints out the shared memory location in the following format: "I am Process A - <pid>". Process A then waits until B and C completes.
2. Process B first checks if the second shared memory location has a "A" in it. When true, it writes its <pid> into the first shared memory location and the string "I am Process B" into the third shared memory location and then signals A & C that it is complete by writing "B" into the second shared memory location (note: B must wait to write into the process identifier shared memory location until after process A writes "A" there). Then process B quits. At this point, process-A prints out the shared memory location in the following format: "I am Process B - <pid>".
3. Process C writes the string "I am Process C" into the string shared memory and its pid into the first shared memory location. It then signals A that it is complete by writing "C" into the second shared memory location (note: C must wait until process B writes "B" into the process identifier shared memory). Then process C quits. At this point, process-A prints out the shared memory location in the following format: "I am Process C - <pid>".
4. Process A needs to keep polling the process identifier shared memory location continuously. After process B writes "I am Process B", process A should be able to print it out. Similarly, after process C writes "I am Process C", process A needs to be able to print it out. Here some ad-hoc synchronization is needed. Specifically, consider using the usleep()

function to delay processes B and C between writing the pid, string and the process identifier to give process A a chance to poll and print. Adjust the delay such that prints happen properly.

5. Process A is the last one to quit and prints out a "GoodBye <pid>" message at the very end before quitting.

Deliverable: THREE different code files; one each for Process A, B and C.

Question 2: Shared Memory (3+3 pts)

Create processes A, B and C from the **same** code file using fork. In this case you will have a single code file where the parent process can serve as ProcessA, and two child processes serve as Process B and C.

Consider TWO implementations here: (i) Process B and C are two separate child processes (using two different forks from the ***parent*** process); (ii) Process B and C are nested forks (here Process C is forked from within ***process B*** and NOT from the parent process).

Deliverable: TWO different code files; one with regular forks, other one with nested forks.

Question 3: Message Queues (9 pts)

Review the programs (spock.c and kirk.c). Answer (or discuss) questions listed below:

a) Discuss and evaluate what happens when you're running both in separate windows and you kill one or the other. (3)

b) Discuss what happens (and why) when you run two copies of kirk. (3)

c) Discuss what happens (and why) when your run two copies of spock. (3)

Deliverable: A pdf document with the answers to (a), (b) and (c).

Question 4: Sort numbers using message queues (15 pts)

Modify the kirk.c and spock.c programs as follows:

(i) kirk.c should send a sequence of numbers to spock.c. spock.c then sorts these numbers in ascending order and prints them out.
(ii) spock.c should still run in an infinite loop to receive inputs from multiple instances of kirk.c running separately.
(iii) Discuss the case when multiple kirk.c execute simultaneously versus each kirk.c instance starts only after the previous one has completed.

Deliverable: Updated kirk.c and spock.c code files and a pdf document discussing your observations in (iii).

Socket programs

Question 5: Sort numbers using socket programming (15 pts)

Modify the client1.cpp and server1.cpp codes to implement a client-server socket program that can sort numbers as explained in Q4.

Note: Multiple clients should be able to send their sequence of numbers to the server simultaneously. The server must "fork" child processes to handle each client separately. Check this link for example code and more explanation: https://www.geeksforgeeks.org/design-a-concurrent-server-for-handling-multiple-clients-using-fork/

Submission:

Upload the relevant files on Canvas.

Late Policy:

Late assignments will incur a penalty of 5 points per day for a maximum of 2 days.