CPSC 457 - Principles Of Operating Systems
Assignment 2
IPC, Dining Philosophers Problem
Fall 2023

1 Objective

The primary objective of this assignment is to deepen students' understanding of inter-process communication (IPC) mechanisms and to introduce the inherent challenges of concurrent programming.

The first section focuses on Inter-Process Communication (IPC). Students are expected to familiarize themselves with various IPC mechanisms. This includes the creation of processes using the fork method, understanding the intricacies of data sharing through shared memory, and establishing communication via message queues.

The second section delves into the realm of Concurrency and Synchronization, using the classic Dining Philosophers problem as a case study. Through this problem, students will be exposed to potential pitfalls in concurrent programming, such as deadlocks and race conditions. Furthermore, students will be tasked with applying synchronization techniques, ensuring that concurrent processes execute both safely and efficiently.

Section 1: Process Creation and IPC Demonstration

You are a software engineer at a tech company that specializes in developing operating system utilities. Your team is working on a project to understand and demonstrate the basics of process creation and Inter-Process Communication (IPC) mechanisms in the operating system. Your manager has assigned you the following tasks:

Part 1: Process Creation with fork()

Your first task is to develop a utility that demonstrates the creation of child processes:

1. Write a program that leverages the fork() system call to spawn child processes.
2. Ensure that both the parent and the child processes print a message indicating their status.
3. Use the return value of fork() to differentiate between the parent and child processes. Your utility should clearly indicate which process (parent or child) is currently executing.

Part 2: IPC using Message Queues

Your manager wants to see a demonstration of communication between processes:

1. Design a program where a parent process communicates with its child process using message queues.
2. The child process should convert the received message to uppercase and relay a response back to the parent process.
3. Ensure that both the parent and child processes display the messages they send and receive for verification purposes.

Part 3: IPC using Shared Memory

To further the project's goals, you are tasked with:

1. Adapting the program from Task 2 to employ shared memory as the communication medium.
2. Structure the shared memory segment to house the message and any necessary flags or indicators for synchronized access.
3. Implement measures to prevent race conditions, ensuring both processes can safely interact with the shared memory.
4. As before, both the parent and child should display the messages they exchange.

Part 4: IPC using Shared Memory with Process Coordination and Synchronization

Your manager emphasizes the importance of process coordination when multiple processes access shared resources. Your final task is:

1. Develop a utility where multiple child processes (minimum of 3) interact with a parent process via shared memory.
2. The shared memory should incorporate a message buffer for child processes to deposit their messages and an array to monitor each child's status (e.g., message pending, message sent).
3. The parent process must periodically scan for new messages from child processes and display them. Upon reading a child's message, the parent should update the child's status to indicate the message has been processed.
4. Integrate synchronization mechanisms (semaphores) to ensure:

- Exclusive write access to shared memory for one child process at a time.
- The parent process waits until a child completes its message before reading.
- A child waits until its message is processed by the parent before writing a new one.

Hint

- Ensure proper error handling for all system calls.
- Remember to clean up any IPC resources (like shared memory segments, pipes, or message queues) after they are used.
- Choose an appropriate synchronization primitive (e.g., semaphores) that fits best for this scenario.
- Ensure that deadlocks and race conditions are prevented.
- Properly initialize, use, and clean up any resources associated with synchronization primitives.
- Child processes should not exit prematurely. Implement a mechanism for the parent to signal all child processes to terminate gracefully once the demonstration is complete.

Execute your code

You can test your code by running it in the Computer Science labs. (cslinux.ucalgary.ca) Make sure that your codes are working in the Computer Science labs as it is the main reference for TAs when grading your work.

Section 2: The Cosmic Conundrum of Concurrent Chopsticks

Problem statement

Far beyond the reach of our telescopes, in the pulsating heart of a distant nebula, floats an elliptical spaceship named "Noodle Nebula". Aboard it, n interstellar astronomers from various dimensions converge to unravel the secrets of the cosmos. The convention is unique: They dine on ethereal noodles while pondering over space-time enigmas.

Each astronomer has an ethereal bowl of noodles placed before them. Suspended in the zero-gravity environment, between every pair of adjacent astronomers, is a quantum chopstick. Unlike ordinary chopsticks, these can only be wielded by one astronomer at a

time. To consume the noodles, an astronomer requires both the quantum chopsticks adjacent to them.

The inter-dimensional astronomers have peculiar habits:

1. If both adjacent chopsticks aren't accessible, the astronomer delves deeper into thoughts about black holes and dark matter, waiting for them to become available.
2. Once done dining, they release the chopsticks, returning to their contemplative state.
3. There are a type of astronomers called *asymmetrical astronomers*, and they pick up the left quantum chopstick, only after they pick up the right one.
4. Their minds are connected via a neural quantum entanglement, ensuring no verbal communication interrupts their profound thoughts or dining.

Devise an algorithm that simulates this multi-dimensional dining conundrum. The challenge is twofold: prevent a cosmic deadlock and ensure no astronomer is left starving in this space-time rendezvous.

Requirement

Your implementation should take account for the following requirements:

- *Symmetric* astronomers, regular type, that pick up both forks at the same time (having at most 0.5s time difference between acquiring the left and right quantum chopsticks is considered symmetric).
- *Asymmetric* astronomers, that pick up only one quantum chopstick at a time. If they manage to pick up one quantum chopstick but not the other, they must release it after a certain time to avoid starvation.
  - An *asymmetric* astronomer must acquire the right quantum chopstick before acquiring the left one (picking up the left fork should be at least 1s after the right one to satisfy the astronomer).
  - If an *asymmetric* astronomer cannot acquire the left quantum chopstick after a certain period, they must release the right quantum chopstick to avoid starvation.
  - Once an *asymmetric* astronomer has acquired both quantum chopsticks, they can eat for a fixed time before putting down the quantum chopsticks.
  - After eating, the astronomer must release both quantum chopsticks.
- If an astronomer could not pick up the quantum chopsticks in its desired way within 2s, they should release the quantum chopstick that they already have.
- After acquiring the quantum chopsticks, each philosopher should eat for a certain amount of time, which is drawn from a random quantum variable with the average of *avg_eat_time*, to account for different eating times.

- Astronomers should be placed around the table in a fairly random manner.
- Deadlocks, a situation where all astronomers await a chopstick and none can proceed, must be circumvented.
- Ensure equitable dining opportunities for all astronomers, preventing any potential starvation.
- Visualize the conundrum by cataloging the state of each astronomer (contemplating or consuming) and the status of each quantum chopstick and print the states in the terminal.
- The design should be universal, catering to any number of n astronomers, not just a finite number.
- Of these n astronomers, a, $a<n$, astronomers are *asymmetric*. Pick a arbitrarily and keep in mind that a should not be larger than n.

Hint

- Explore the potential of quantum semaphores or similar synchronization entities to guard the chopsticks.
- Reflect on the sequence of acquiring and releasing chopsticks.
- Ponder on a stratagem where not all astronomers can attempt to seize chopsticks simultaneously.

Bonus: A Cosmic Dining Experience milky-way chopsticks

Oh! We forgot to tell you about a completely different type of dining astronomers.

- *Greedy* astronomers must eat two times, drawn from a random quantum variable with average of 2**avg_eat_time*, in a row before releasing the quantum chopsticks.
  - Non-greedy astronomers follow the standard rules: they pick up quantum chopsticks, eat for a fixed time, and then release the quantum chopsticks.
  - *Greedy* astronomers, after eating twice, must relinquish both quantum chopsticks simultaneously.
  - The goal is to maximize overall throughput and ensure that no astronomer starves while accommodating the *greedy* astronomers' preference for consecutive meals.

Let's be inclusive of all these types of astronomers and implement a scenario where all three types of astronomers dine together. Now, we have a cosmic feast for these astronomers, whether they are *symmetric*, *asymmetric*, or *greedy*!

2 Grading

The entire assignment carries 100 points. Section 1 will carry 30 out of the 100 points (part 1 is worth 6 points and parts 2-4 are worth 8 points each). The first part of Section 2, dining astronomers with only *symmetric* types will carry 35 points, and the solution accompanying both *symmetric* and *asymmetric* astronomers will carry 35 points out of the 100. And for the bonus part, *A Cosmic dining experience*, you will receive 20 extra points.

3 Deadline

The assignment is due by 10th November at 11:59 PM, after that the deduction policy of 20% daily (part of the day is considered as a full day) will be applied. The first 15 minutes of the penalty day is a grace period and will not result in a deduction.

4 Submission

To submit, create a new folder called "assigment2" and place all files you wish to submit in this folder just like the tree you are seeing below. Use the command "tar -zcvf zip_name.tar.gz assigment2" and upload the zip_name.tar.gz. The Zip File name should be the student name+ID (example: Name = John Doe, Student ID = 12345678, FILENAME = john_doe_12345678). Ensure that you have added all the files while zipping. The answer template will be uploaded on D2L.