University of Calgary
Faculty of Science
Department of Computer Science

CPSC 457 - Principles Of Operating Systems
Assignment 1
Shell Scripting and Multiple Producer-Consumer Problem
Fall 2023

Section 1: Shell Scripts

1.1 Objectives

The following exercises in this part cover essential shell scripting concepts, including conditional statements, pipes, file and directory operations, arithmetic calculations, and the use of commands like "find" and "time" By completing these tasks, you will gain hands-on experience and a deeper understanding of how to leverage shell scripts for everyday problem-solving.

1.2 Background

In this background section, we explored the world of shell scripting, focusing on the fundamental aspects of what shell is, where it's used, why it's a crucial tool and a captivating fun fact related to shell scripting.

What is Shell? Shell is a command-line interface that acts as a bridge between users and the underlying components of a computer's operating system. It enables users to interact with the system through text-based commands.

Where it's Used? Shell environments are integral to Unix-like operating systems like Linux and macOS, and they also exist in various forms in Windows systems. Shell scripting is essential for system administrators, developers, and anyone looking to maximize their computer's capabilities.

Why it's Used? Shell scripting is employed for a range of purposes, including automation of tasks, customization of system settings, batch processing, resource management, and real-time interactivity. It provides a powerful means of improving efficiency and productivity.

As a fun fact about Shell scripting, the term "shell" is aptly chosen, as it encapsulates and protects the inner workings of the operating system, much like a protective shell surrounds an organism.

1.3 Questions

1. Write a shell program to calculate if a number is even or odd.
2. Write a shell program to list the number of files and directories along with their permissions in a particular directory.
3. Write a shell program to calculate the sum of n numbers. (n should be greater than 3)
4. Write a shell script that calculates and displays the total size of all files in a directory and 3 levels of its subdirectories using the "find" command and pipelining. Display the size in human-readable format (e.g., KB, MB, GB)
5. Write a shell program to determine whether the input is Armstrong number or not. (An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since 3**3 + 7**3 + 1**3 equals 371.)

1.4 How to Do It

1.4.1 Choose a text editor

You can use "Vi", "Vim", "nano", or "gedit" as editors available in the terminal. Also, you can use "VSCode", "Atom", or "Sublime" which all support GUI.

1.4.2 Write your code

Write the code that you want to execute in bash language. Please use meaningful names for the variables and functions and write comments whenever it is needed.

1.4.3 Make your code executable

Use "chmod" command to make your code executable. Please note that your code won't work if you skip this step.

1.4.4 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: Producer-Consumer Problem

2.1 Objectives

Processes, self-contained program execution units that have their own memory space, registers, and program counter, are designed to achieve a specific purpose that is assigned to them. Different processes achieve different purposes. However, oftentimes, more than one process may need to access the same resource or exchange data with another process. This leads us to the concept of coordinating and managing access to shared resources amongst different processes in a manner where they do not interfere with each other in undesirable ways; this is called *Synchronization*. Your assignment is to make use of the basic binary and counting semaphores discussed in class in order to simulate a busy day at a carnival thus implementing a classic synchronization problem - The Producer Consumer Problem.

2.2 Background

One of the most famous and essential problems in synchronization and concurrency is the *producer-consumer* problem. Our focus is on a specialized version of this problem called the *bounded-buffer* problem. In such a problem, we have two processes: one called the *producer* and the other the *consumer*, and additionally, there is a *bounded buffer* that both the producer and consumer write to and read from. The producer is in charge of adding items to the bounded buffer, and the consumer is in charge of removing them. If the buffer is full, the producer waits until a consumer takes something. If the bounded buffer is empty, the consumer waits until the producer adds something.

In this assignment, we will implement an extended form of the producer-consumer problem: the *multiple-producer multiple-consumer* problem. It is identical to the *consumer-producer* problem, except as the name implies, multiple processes can produce, and multiple processes can consume. Also, there may be more than one bounded buffer.

You will solve this problem in the context of a busy day at the carnival, where you have two producer processes (balloon artists), three consumer processes (customers), and two bounded buffers (balloon carts).

Scenario

You are the owner of Balloon Brigade, a company that sells different types of balloon figures and you have been asked to set up a stall at the Calgary Carnival. However, the organizers are adamant that you can't bring ready-made balloon figures and sell them, but rather, your balloon artists should make the figures at the carnival and put them in carts throughout the duration of the carnival as they believe that many people enjoy watching how these figures are made.

You will thus be in charge of organizing your two specialized balloon artists (represented by two producer processes) to constantly resupply two carts of balloon figures (represented by two bounded buffers), for three queues of excited customers to take from (represented by three consumer processes). Each artist specializes in 2 figures, and they strictly add items only to their respective cart (buffer):

1. Balloon Bob specializes in balloon animals:
   (a) Balloon Monkey
   (b) Balloon Dog
2. Helium Harry specializes in balloon houses:
   (a) Balloon Hut
   (b) Balloon Tower

A balloon artist should randomly pick between either of their figures, and add it to the balloon cart at a random rate between 1 and 10 seconds (inclusive).

The three consumer processes are each programmed as an infinite loop() of customers:

1. The first process will represent customers that want any animal balloon.
2. The second process will represent customers that want any house balloon.
3. The third process will represent customers that want one of each.

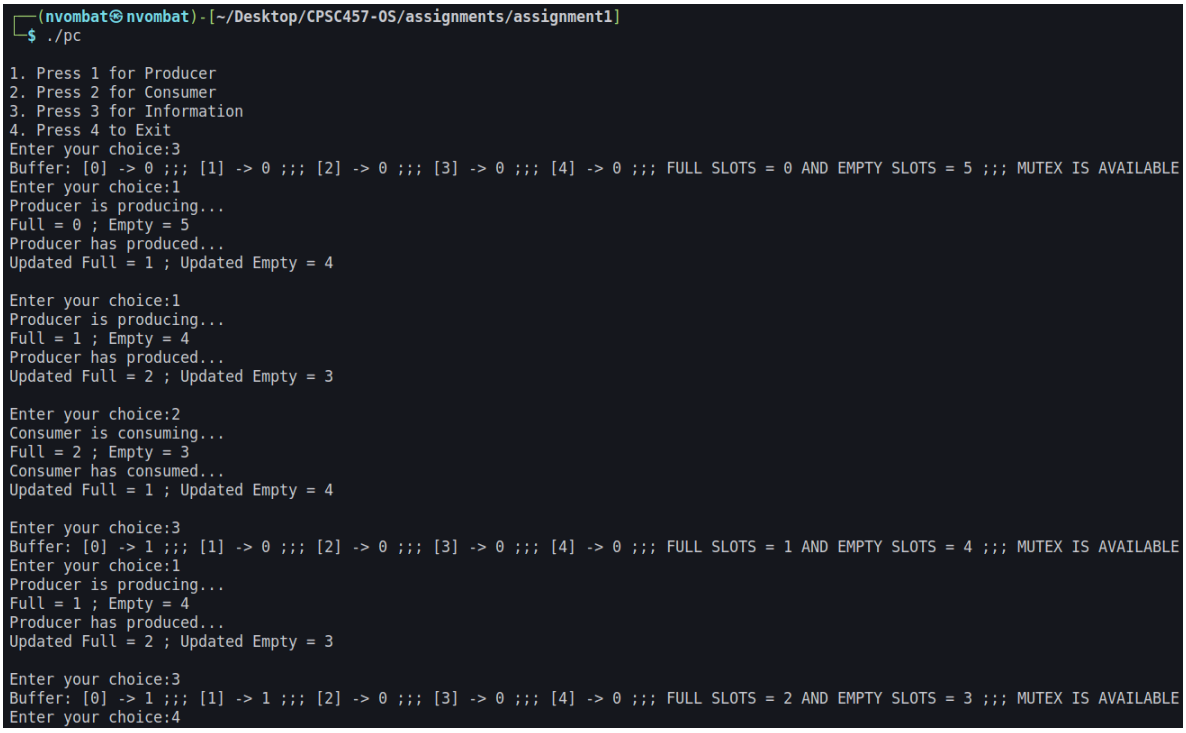
A customer needs to wait if there's not enough of any of their desired balloon figures. The balloon artists need to wait if any of their respective carts are full. After a customer takes an item, the respective customer process should wait for a random amount of time between 5 and 15 seconds seconds (inclusive) before iterating again.

2.3 Questions

Part 1 - Your stall at the carnival has just ONE balloon artist, Balloon Bob (producer) as Helium Harry is running late. Balloon Bob is making balloon animals and putting them into a cart (bounded buffer). Customers are lined up to pick up these balloon animals (consumer).

Part 2 - Helium Harry has just arrived with his cart. Now there are TWO balloon artists (two producers). They put their respective balloons into their respective carts (two bounded buffers). There are three types of consumers - people who want balloon animals, people who want balloon houses and people who want both animals and houses.

Simulate these scenarios the best you can. For part 1, make it such that the user can interact with the program. The following screenshot shows an example output for part 1 of your assignment:



For part 2, ensure that you display your understanding of multi-threading by simulating an environment where producers and consumers are able to do their tasks without any errors. Make sure the simulation runs for 45 seconds.

For both part 1 and 2, ensure you have an option which displays the contents of the buffer and any other information regarding its state.

2.4 How to Do It

Before beginning part one, you need to understand how the basic locking mechanism works between processes and how to synchronize them such that there are no *race conditions* between them, which occur when two or more processes want to alter the shared data at the same time.
The following steps should help you organize your work on the assignment.

2.4.1 Create and initialize the buffers and semaphores

For each bounded-buffer, we need a **binary semaphore** that makes sure only one process can write to the buffer at any one time. Call this semaphore *mutex* as it is acting as a lock for the buffer. For each buffer, we will also need two **counting semaphores** - called *full*

and *empty* - that ensure a producer can't go into its critical section if the buffer is full, and likewise, a consumer can't do so if their respective buffer is empty.

Play with these semaphores to make sure mutual exclusion is being met. *Hypothetically*: If you were to use just 1 buffer, to keep track of the buffer and its state, you would have needed to use an integer array that stores values 0,1,2. This will allow the buffer to be managed easily as 0 will represent an empty position, 1 will represent balloon animals and 2 will represent balloon houses.

2.4.2 Create the producers and consumers

You should now have the necessary setup to start working on your producer and consumer processes to simulate the fun-filled carnival. Each time a producer or consumer executes its critical section (i.e. adds/takes from the buffer), they should print the following:

1. Whether it is a producer or consumer doing the action
2. Which producer/consumer it is:
   (a) Balloon Bob vs Helium Harry for the producers
   (b) Balloon Animals vs Balloon Houses vs hybrid for the consumers
3. Which variation of the balloon figures the producer/consumer has added/taken from the buffer

Keep in mind that you should be implementing bounded-buffers! This means utilizing and keeping track of the **in** and **out** positions of each buffer. Simply adding/taking from any empty/full slot will not receive credit!

2.4.3 Create and Use Threads

In this part you need to re-implement the problem using threads. From an educational point of view, this should reflect your understanding of the use of threads vs processes, and how to handle their side effects, if any. In your implementation, you need to consider the following:

1. Avoid zombie processes and threads (**pthread_exit** and **pthread_join**, etc.)
2. Avoid multiple processes writing to the same shared memory at the same time thus avoiding data corruption (race conditions)

3 Grading

The entire assignment carries 100 points. Section 1 (shell programs) will carry 30 out of the 100 points (questions 1-3 are worth 5 points each and questions 4 and 5 are worth 7.5 points each). The first part of Section 2 will carry 35 points and the second part of Section 2 will carry 35 points out of the 100.

4 Deadline

The assignment is due by October 6 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 deduction.

5 Submission

To submit, create a new folder called "dir_name" and place all files you wish to submit in this folder. Use the command "tar -zcvf zip_name.tar.gz dir_name" 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.