Operating Systems - Assignment 3 - Monsoon 2022

Arani Bhattacharya, Sambuddho Chakravarty

November 26, 2022

Deadline: December 20, 2022

Total points: 150

1 Modified Dining Philosophers Problem (60 points)

The dining philosophers problem contains five philosophers sitting on a round table can perform only one among two actions – eat and think. For eating, each of them requires two forks, one kept beside each person. Typically, allowing unrestricted access to the forks may result in a deadlock. (a) Write a program to simulate the philosophers using threads, and the forks using global variables. Resolve the deadlock using the following techniques:

1. Strict ordering of resource requests, and
2. Utilization of semaphores to access the resources.

(b) Repeat the above system only using semaphores now with a system that also has two sauce bowls. The user would require access to one of the two sauce bowls to eat, and can access any one of them at any point of time.

What to submit/rubric.

1. Creating the threads and shared variables [**5 * 2 = 10 points**].
2. Two variants of the program (a) [**10 * 2 = 20 points**]. Full compilation and correct functionality of all the programs (20 points). Program compiles successfully but doesn't meet all the functionality requirements (15 points). Program doesn't compile, even if program logic seems apparently correct (0 points).
3. Two variants of the program (b) [**10 * 2 = 20 points**]. Full compilation and correct functionality of all the programs (20 points). Program compiles successfully but doesn't meet all the functionality requirements (15 points). Program doesn't compile, even if program logic seems apparently correct (0 points).
4. A proper makefile building all the source files [**5 points**].

5. A readme file with a short explanation of the resources and primitives used [**5 points**].

2 Interprocess Communication (50 points)

Write two programs P1 and P2. The first program P1 needs to generate an array of 50 random strings (of characters) of fixed length each. P1 then sends a group of **five** consecutive elements of the array of strings to P2 along with the ID's of the strings, where the ID is the index of the array corresponding to the string. The second program P2 needs to accept the received strings, and send back the **highest** ID received back to P1 to acknowledge the strings received. The program P2 simply prints the ID's and the strings on the console. On receiving the acknowledged packet, P1 sends the next five strings, with the string elements starting from the successor of the acknowledged ID.

The above mechanism needs to be implemented using three different techniques: (i) Unix domain sockets, (ii) FIFOs, and (iii) shared memory. Please note that you should NOT make assumptions about the reliability of the interprocess communication mechanism, unless they are guaranteed by the mechanism itself. You should also not use redundant mechanisms to guarantee reliability if the protocol itself guarantees it. Print the amount of time required to finish receiving the acknowledgment of all 50 strings in the three cases.

What to submit/rubric.

1. Three variants of the program P1 (one each for communicating using Unix domain sockets, FIFOs and shared memory respectively) [**5 * 3 = 15 points**]. Full compilation and correct functionality of all the programs (15 points). Program compiles successfully but doesn't meet all the functionality requirements (8 points). Program doesn't compile, even if program logic seems apparently correct (0 points).

2. Three variants of the program P2 (one each for communicating using Unix domain sockets, FIFOs and shared memory respectively) [**5 * 3 = 15 points**]. Full compilation and correct functionality of all the programs (15 points). Program compiles successfully but doesn't meet all the functionality requirements (8 points). Program doesn't compile, even if program logic seems apparently correct (0 points).

3. A proper makefile building all the source files [**5 points**].

4. A readme file with a short explanation of the resources and primitives used [**5 points**]. Makefile and readme will be checked only if the source files compile successfully.

3 Kernel Module (50 points)

This exercise requires you to write your own small kernel module. You require to implement a kernel system call as a module. The task of the system call would be to read the entries of the process `task_struct` corresponding to any given process (supplied as input via command line argument) and prints the

values of the following field: pid, user_id, process group id (pgid) and command path. The system call should be implemented in the kernel (and not directly as a mainline kernel function, like that was done for A2). It should be functional only when the module is loaded, not otherwise.

What to submit/rubric.

1. Fully functional system call that runs only when the module is loaded, and not otherwise **[40 points]**. [Module that otherwise seems correctly written but doesn't run as expected (nevertheless runs and prints garbage output) **[20 points]**. — Module that doesn't compile but seems to be having a the correct logic**[20 points]**

2. A proper Makefile to compile the module **[5 points]**.

3. A readme file describing the program logic **[5 points]**.