OS Assignment 3

05.11.2022

Process Synchronization Problems

In this assignment, you will implement solutions to some of the synchronization problems. You should implement the solutions in C language using pthread library. Submission of the files must contain the following format: rollno-3-qn.c, where rollno is your roll number and qn is the question number (eg. 1, or 2). All the files must be zipped to a single file with the name rollno-assign3.zip. Any other file formats and name formats will be rejected.You have to submit your file in the link provided in eduserver course page.

Maximum weight-age will 10 marks. Any late submissions will lead to reduction in marks.

Problem 1

In this question, you will implement a multi-threaded program where the parent with shared *counter* variable invokes two child threads which will run concurrently. The first thread increments a shared variable *counter* 10^7 times. The second thread will decrements the *counter* 10^7 times. After both threads are done, the parent should print the value of the *counter*. You should use semaphore library functions available in pthread library so that the value printed by the parent is zero.

Problem 2

In this problem, you will implement a solution to the dining philosopher's problem. You should implement the solution in a multi-threaded way where a parent process spawns five child threads which will run concurrently to simulate the behaviour of five philosophers. There should be one fork between two adjacent philosophers. Each philosopher thinks for a random amount of time and then feels hungry. In order to eat, a hungry philosopher requires forks on two sides of him. If he can grab those, he eats for some time and then leaves back the forks from where he has taken those and restarts thinking.

Your implementation must have two parts to simulate the deadlock situation and the deadlock-free situation. When all the philosophers have grabbed their

left forks, and are waiting for their right neighbors to release the right forks, the deadlock will happen. You may implement any solution to remove deadlock using semaphores for synchronization and mutual exclusion. Maintain a shared data structure to keep track of fork usage. The parent process periodically checks for a deadlock (cycle) in the shared data structure. Suitable diagnostic messages must be printed at various stages of your program execution. Sample messages are given below.

Philosopher 3 starts thinking
Philosopher 2 starts eating
Philosopher 0 grabs fork 0 (left)
Philosopher 4 ends eating and releases forks 4 (left) and 0 (right)
The parent detects a deadlock, going to initiate the recovery