Operating Systems Assignment 4

Deadline : 08 October, 2016 11:55 PM

1. Producer Consumer Problem:-

- Given the size of a circular buffer and number of readers, write a code with n+1 threads, n for readers and 1 for the writers. All are accessing the same circular buffer. The following conditions should be satisfied:
  - A reader cannot read in an empty location.
  - A reader will not read from the same location twice if the writer has not overwritten.
  - A writer cannot overwrite a place which has not been read by every reader.
- Input : Size of buffer, n. You can use random function for writing data into the buffer.

2. Concurrent Merge Sort:-

- Given a number n and n numbers, sort the numbers using Merge Sort.
- Recursively make two child processes, one for the left half, one of the right half. If the number of elements in the array for a process is less than 5, perform a selection sort.
- The parent of the two children then merges the result and returns back to the parent and so on.
- Compare the performance of the merge sort with a normal merge sort implementation and make a Report

Note:- You must use the shmget, shmat functions as taught in the tutorial.

3. The queue at the Polling booth:-

- People are fed up with waiting at polling booth on day of election. So, government has decided to improve its efficiency by automating the waiting process. From now on, voters will be robots and there will be more than one EVM at each booth. Each of these EVM's can now accept vote from more than one person at a time by having different number of entries for voting. However one person can only vote once. Each robot and each EVM is controlled by a thread. You have been hired to write synchronization functions that will guarantee orderly use of EVM's. You must define a structure struct booth, plus several functions described below.
- When an EVM is free and is ready to be used it invokes the function **polling_ready_evm(struct booth *booth, int count)** where count indicates how many entries are available to vote at this instant. The function must not return until the EVM is satisfactorily allocated (all voters are in their entry , and either the EVM is full or all waiting voters have been assigned).

- When a voter robot arrives at the booth, it first invokes the function **voter_wait_for_evm(struct booth *booth)**. This function must not return until an EVM is available in the booth (i.e., a call to polling_ready_evm is in progress) and there are enough free slots on the evm for this voter to vote. (One slot per one voter). Once this function returns, the voter robot will move the voter to the assigned slot (you do not need to worry about how this mechanism works).
- Once the voter enters the slot, he/she will call the function **voter_in_slot(struct booth *booth)** to let the evm know that they reached the slot.
- Create a file that contains a declaration for struct booth and defines the three functions above, plus the function booth_init, which will be invoked to initialize the booth object when system boots. In addition:
    - No Semaphores should be used for solving this one.
    - You should not use more than a single lock in each struct booth.
    - You may assume that there is never more than one EVM in the booth available free at once, and that any voter can vote in any EVM.
    - Your code must allow multiple Voters to board simultaneously (it must be possible for several Voters to have called **voter_wait_for_evm**, and for that function to have returned for each of the Voters, before any of the Voters calls **voter_in_slot**).
    - Your code must not result in busy-waiting (Deadlocks).

Upload Guidelines :

- Format: roll_number_assignment4.zip
- Write a Makefile to compile your files. Include a Readme containing the details of how to run your code.
- For questions 1 and 3 write a brief report containing the details of your implementation saying how did you solve the problem.
- **DO NOT COPY**. More emphasis is on how you tackle the situation. If your code works and you are unable to explain it, you will end up getting very less marks.