Assignment 5

Creative Design Thinking Problems on Operating Systems

Topics Covered: Operating Systems Fundamentals, Programming, POSIX Threads, Process Scheduling, Process Synchronization, Distributed Clocks, Shared Memory, Deadlocks, Starvation, Software Testing, Software Verification and Validation

Notes:

a. Your creative design thinking process should coincide with Distributed Operating Systems.
b. You can use C, C++ and Java as a language while you want to code the problems.
c. If you feel about the randomization then you have to use only srand(). If you had any feeling regarding the non-preemption and preemption, then use only round robin scheduling with preemption.
d. Take all the assumptions whenever and wherever it is necessary and just record them in a file.
e. Use the following or equivalent code for implementing the Clock.
f. 10 points are for creative thinking.

Clock:

Your code should use the chrono library's high_resolution_clock class. The following example of how to do this in nanoseconds is found on Stack Overflow.

```
#include <iostream>
#include <chrono>
typedef std::chrono::high_resolution_clock Clock;

int main()
{
    auto t1 = Clock::now();
    auto t2 = Clock::now();
  std::cout << "Delta t2-t1: "
        << std::chrono::duration_cast<std::chrono::nanoseconds>(t2 - t1).count()
        << " nanoseconds" << std::endl;
}
```

1. A barbershop consists of a waiting room with *n* chairs, and the barber room containing the barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves

the shop. If the barber is busy, but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber.

a. Write an algorithm/pseudocode as a proof of your work.
b. Using POSIX threads, mutex locks, and semaphores write a program to coordinate the barber and the customers including clocks keeping in the mind that you must adhere to happens-before relationship according to the Lamport's Vector Logical Clock.
c. Provide the test suite and the results of the Alpha Testing.
d. Provide the test suite and the results of the Beta Testing.

Point(s) [2+6+1+1]

2. Consider a system with three smoker processes and one agent process. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker needs three ingredients: tobacco, paper, and matches. One of the smoker processes has paper, another has tobacco, and the third has matches. The agent has an infinite supply of all three materials. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient then makes and smokes a cigarette, signaling the agent on completion. The agent then puts out another two of the three ingredients, and the cycle repeats.
    a. Write an algorithm/pseudocode as a proof of your work.
    b. Using POSIX threads, monitors, and semaphores write a program to synchronize the agent and the smokers including clocks keeping in the mind that you must adhere to happens-before relationship according to the Lamport's Vector Logical Clock.
    c. Provide the test suite and the results of the Alpha Testing.
    d. Provide the test suite and the results of the Beta Testing.

Point(s) [2+6+1+1]

3. A university computer science department has a teaching assistant (TA) who helps undergraduate students with their programming assignments during regular office hours. The TA's office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the TA is currently helping another student. When there are no students who need help during office hours, the TA sits at the desk and takes a nap. If a student arrives during office hours and finds the TA sleeping, the student must awaken the TA to ask for help. If a student arrives and finds the TA currently helping another student, the student sits on one of the chairs in the hallway and waits. If no chairs are available, the student will come back later.
    a. Write an algorithm/pseudocode as a proof of your work.
    b. Using POSIX threads, mutex locks, and semaphores, implement a solution that coordinates the activities of the TA and the students including clocks keeping in

the mind that you must adhere to happens-before relationship according to the Lamport's Vector Logical Clock.

c. Provide the test suite and the results of the Alpha Testing.
d. Provide the test suite and the results of the Beta Testing.

Point(s) [2+6+1+1]

4. The readers-writers problem is a classical one in computer science: we have a resource (e.g., a database) that can be accessed by readers, who do not modify the resource, and writers, who can modify the resource. When a writer is modifying the resource, no-one else (reader or writer) can access it at the same time: another writer could corrupt the resource, and another reader could read a partially modified (thus possibly inconsistent) value. Consider the three variants of the existing problem.
    a. Give readers priority: when there is at least one reader currently accessing the resource, allow new readers to access it as well. This can cause starvation if there are writers waiting to modify the resource and new readers arrive all the time, as the writers will never be granted access if there is at least one active reader.
    b. Give writers priority: here, readers may starve.
    c. Give neither priority, "the third readers-writers problem": all readers and writers will be granted access to the resource in their order of arrival. If a writer arrives while readers are accessing the resource, it will wait until those readers free the resource, and then modify it. New readers arriving in the meantime will have to wait.
        i. Write an algorithm/pseudocode as a proof of your work.
        ii. Using POSIX threads, mutex locks, and semaphores write a program for the third readers-writers problem including clocks keeping in the mind that you must adhere to happens-before relationship according to the Lamport's Vector Logical Clock.
        iii. Provide the test suite and the results of the Alpha Testing.
        iv. Provide the test suite and the results of the Beta Testing.

Point(s) [4+12+2+2]

5. A single-lane bridge connects the two Vermont villages of North Turnbridge and South Turnbridge. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if a northbound and a southbound farmer get on the bridge at the same time. (Vermont farmers are stubborn and are unable to back up.)
    a. Using semaphores and/or mutex locks, design an algorithm in pseudocode that prevents deadlock. Initially, do not be concerned about starvation (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa).
    b. Modify your solution of 5 so that it is starvation-free.

c. Implement your solution of 5 using POSIX synchronization. Represent northbound and southbound farmers as separate threads. Once a farmer is on the bridge, the associated thread will sleep for a random period, representing traveling across the bridge. Design your program so that you can create several threads representing the northbound and southbound farmers.

Point(s) [6+4+10]

6. Some monkeys are trying to cross a ravine. A single rope traverses the ravine, and monkeys can cross hand-over-hand. Up to five monkeys can be hanging on the rope at any one time. if there are more than five, then the rope will break, and they will all die. Also, if eastward-moving monkeys encounter westward moving monkeys, all will fall off and die.

   Each monkey operates in a separate thread, which executes the code below:

```
typedef enum {EAST, WEST} Destination;

void monkey(int id, Destination dest) {
    WaitUntilSafeToCross(dest);
    CrossRavine(id, dest);
    DoneWithCrossing(dest);
}
```

   Variable id holds a unique number identifying that monkey. CrossRavine(int monkeyid, Destination d) is a synchronous call provided to you, and it returns when the calling monkey has safely reached its destination. Use semaphores to implement WaitUntilSafeToCross(Destination d) and DoneWithCrossing(Destination d).

   a. For Part (a) You implementation should ensure that:
      i. At most 5 monkeys simultaneously execute CrossRavine().
      ii. All monkeys executing in CrossRavine() are heading in the same direction.
      iii. No monkey should wait unnecessarily. (This way, you can maximize the throughput)

   b. Instead of maximizing the throughput, your solution for Part (b) should ensure there is no starvation (the first two conditions in part (a) still apply). That is, no monkey should wait at the ravine forever. You can assume that semaphore's wait queue is FIFO queue.

c. Now instead of monkeys, we have students crossing the ravine. Because students are much smarter than monkeys, assume that students heading different directions CAN be on the rope at same time. Write a semaphore implementation that ensures the following requirements.
    i. At most M eastward-moving students simultaneously execute CrossRavine()
    ii. At most N westward-moving students simultaneously execute CrossRavine()
    iii. At most K students simultaneously execute CrossRavine()
    iv. you solution should maximize the throughput under above conditions.

Point(s) [6+6+8]

Submission:

Submit all source code files and any required data files in a zip file. Include a write up as a .pdf including:

- Instructions for compiling and running the program including any files or folders that must exist.
- Record the results of alpha and beta testing.

Submission should be submitted via Canvas.