CS4532 Concurrent Programming

Homework 1 and 2

Due – June 21 before 11:55 PM

Answers can be submitted to LMS as a .pdf or can hand them over during the class before the deadline.

1. Discuss whether the following code satisfy the safetyness and liveness properties? [4 marks]

```
class Account {
  double balance;
  int id;

  void withdraw(double amount){
     balance -= amount;
  }

  void deposit(double amount){
     balance += amount;
  }

   void transfer(Account from, Account to, double amount){
        sync(from);
        sync(to);
           from.withdraw(amount);
           to.deposit(amount);
        release(to);
        release(from);
    }
}
```

2. Consider a computer that does not have a TSL instruction but does have an instruction to swap (SWP) the contents of a register and a memory word in a single indivisible action. Use it to write alternative routines for *enter_region* and *leave_region* such as the one found below.

[3 marks]

```
enter_region:
       TSL REGISTER,LOCK       | copy lock to register and set lock to 1
       CMP REGISTER,#0         | was lock zero?
       JNE enter_region        | if it was non zero, lock was set, so loop
       RET                     | return to caller; critical region entered
leave_region:
       MOVE LOCK,#0            | store a 0 in lock
       RET                     | return to caller
```

3. A group of students are studying for CS 4532 exam. The students can study only while eating pizza. Each student executes the following loop:

```
while (true) {
   pick up a slice of pizza;
    study while eating the pizza
}
```

If a student finds that the pizza is gone, the student goes to sleep until another pizza arrives. The first student to discover that the group is out of pizza calls Kamal's Pizza to order another pizza before going to sleep. Each pizza has s slices. Once Kamal delivers pizza, he wake up all the students in the group. Then the students pick up a slice of pizza and go back to studying, and the process continues.

Write code to synchronize the student threads and the Kamal's pizza delivery thread.

Your solution should avoid deadlock and call Kamal's Pizza (i.e., wake up the delivery thread) exactly once each time a pizza is exhausted. No slice/piece of pizza may be consumed by more than one student. Comment your code. [6 marks]

4. Write programs to solve the following problem. Pseudo codes are sufficient. Clearly state any assumptions made. [4x2 marks]

    a. Tickets for a concert are to be sold through a web site. Only a limited number of tickets are available and multiple fans may try to purchase them at the same time. Each fan can purchase up to 4 tickets at a time. Write a server-side script to keep track of the remaining number of tickets and income earned from tickets. No need to keep track of who purchased tickets.

    b. Kamal and Rani work in a restaurant that specialises in a made-to-order Koththu Roti and Curry combination. Kamal makes 2 rounds of Koththu Roti while, at the same time, Rani makes 2 bowls of Curry. When the Koththu Roti and Curry are ready Kamal and Rani each, at the same time, serve a Koththu Roti and Curry combination to a customer(s). These actions are repeated throughout the day. Note that Kamal cannot serve until Rani has made the curry and Rani cannot serve until Kamal has made the Koththu Roti. Write a concurrent program that demonstrates the behavior of Kamal and Rani while satisfying the given constraints.

5. A resource allocation grid shows *illegal*, *safe*, and *unsafe* states for a sequence of requests (REQ) and releases (REL) of resources for 2 processes. Draw a resource allocation grid (like the one in slide 15, lesson 09 – Deadlocks) and clearly identify the *illegal* (forbidden), *safe*, and *unsafe* states (assume the Banker's algorithm is used where these states could lead to an illegal state) for 2 processes, P_1 and P_2, with the following list of requests. All resources are unshareable. State any assumptions you make. [4 marks]

    P_1: REQ{c, e}, REL{c}, REQ{b, d}, REL{e}, REL{d}, REL{b}

    P_2: REQ{a, e}, REL{e}, REQ{b}, REQ{c}, REL{a, b}, REL{c}