CSCE 451/851 Homework 2

Assigned: Feb 29, 2020
Due: Mar 15, 2020 11:59 p.m.
Submit: Upload to Canvas as PDF
100 points total

1. **[25 points]** Assuming that the instruction `XCHG(int *var1, int *var2)` can be used to atomically swap two integer pointer variables (e.g. before `XCHG: *y = 0` and `*z = 1` and after `XCHG: *y = 1` and `*z = 0`), write the bodies of `lock()` and `unlock()` to illustrate how the XCHG can be used to provide mutual exclusion. Assume multiple threads can access `increment(void)` and `decrement(void)`. Note that `increment(void)` and `decrement(void)` just modify the shared value `shared` but their precise purpose is irrelevant.

```
#define N 10000
int bolt = 0;
int shared = 0;

void lock(int * key);
void unlock(int * key);

void increment(void)
{
    int key = 1;
    while(1) {
        lock(&key);
        shared = shared + N;
        unlock(&key);
    }
}
void decrement(void)
{
    int key = 1;
    while(1) {
        lock(&key);
        shared = shared - N;
        unlock(&key);
    }
}
```

[15 points]

```
void lock(int * key)
{
    // you can access the actual value of key by using *key




}
```

[10 points]

```
void unlock(int * key)
{
    // you can access the actual value of key by using *key




}
```

2. **[10 points]** In a mailbox based message passing system, a process is not put in a waiting queue but receives an error signal upon sending to a full mailbox. Similarly, a process is not put in a waiting queue but receives an error signal upon trying to receive from an empty mailbox. Then, the process will try its sending or receiving operation repetitively until it succeeds. Does this approach suffer from race conditions? Why?
3. **[25 points]** Jurassic Park consists of a dinosaur museum and a park for safari riding. There are `k` single-passenger cars and `p` passengers. Passengers wander around the museum for a while, then line up to take a ride in a safari car. When a car is available, it loads a passenger and rides around the park for a random amount of time. If the `k` cars are all out riding passengers around, then a passenger who wants to ride waits; if a car is ready to load but there are no waiting passengers, then the car waits. Solve this problem using semaphores.
   [5 points]

```
// declare global variables here (eg. semaphores, global counters etc)
```

   [10 points]

```
void CarThread(int i)
{




}
```

   [10 points]

```
void PassengerThread(int i)
{




}
void main()
{
    thread_begin(PassengerThread(1), PassengerThread(2), ...,
        PassengerThread(p), CarThread(1), CarThread(2), ..., CarThread(k));
}
```

4. **[40 points]** The Bear and the Honeybees. There are `n` honeybees and a bear. They share a pot of honey. The pot is initially empty; its capacity is `H` equal-portions of honey. The bear sleeps until the pot is full, then eats the entire pot of honey and goes back to sleep. Each bee repeatedly gathers one portion of honey and puts it in the pot; the bee that fills the pot full awakens the bear. Note that while the bear is eating honey, no honeybees can add any honey to the pot.

Represent the bear and honeybees as processes and develop pseudocode that simulates the actions of the bear and honeybees. Use semaphores for synchronization. Show the declarations and initial values of all semaphores that you use (e.g., semaphore mutex = 1). In your pseudocode use `semWait(sem)` and `semSignal(sem)` operations to operate semaphores.
[10 points]

```
// global variables declarations
```

[15 points]

```
void bear(void)
{









}
```

[15 points]

```
void honeybee(void)
{









}
```