CSE-381: Systems 2

Due: Wednesday October 30 2019 before 11:59 PM

Email-based help Cutoff: 5:00 PM on Tue, Oct 29 2019

Homework #7

Maximum Points: 17

Submission Instructions

This homework assignment must be turned-in electronically via Canvas. First save this document using your MU ID (example: `raodm_hw7.docx`). Next, type in your responses to each question (right after the question in the space provided) in this MS-Word document. You may use as much space as you need to respond to a given question. Once you have completed the assignment upload the document back to Canvas.

Note that copy-pasting from electronic resources is plagiarism. Consequently, you must suitably paraphrase the material in your own words when answering the following questions.

Name: Ce Zhang

Objective

The objective of this part of the homework is to review and recapitulate (in preparation for Exam #2) various concepts related:

- Threads
- Race conditions
- Critical Sections
- Semaphores and Monitors

Required Reading

First review the following contents related to this homework exercise prior to proceeding with the homework

1. Review slides on Threads and Synchronization off Canvas.
2. Review Chapter 4 and Chapter 6 from the reference e-book "Operating System Concepts" (Link available off Syllabus page in Canvas) prior to proceeding with this exercise.

1. What is a race condition? Explain in sufficient detail on how to race conditions typically manifest (or how race conditions are observed by an user) themselves in programs? [**1 point**]

   Race condition is the term used to denote inconsistent operation of multi-process/multi-threaded programs.
   The program runs correct in most time. occasionally, the program does not run correctly

2. Complete the body of `main` method (without modifying any other part of the program) such that the C++ program will suffer from a potential race condition [**2 points**]

```
#include <iostream>
#include <thread>

void threadMain(std::string& str) {
   // Convert all characters
   for (char& c : str) {
     c++;
   }
}

int main() {
        const int THREAD_COUNT = 50;
        std::vector<std::thread> threadGroup;
        for (inti= 0; (i< THREAD_COUNT); i++) {
            threadGroup.push_back(std::thread(threadMain));
        }
        for (auto&t : threadGroup) {
             t.join();
        }
        std::cout<< "Value of num = "<< num << std::endl;



        return 0;
}
```

   Briefly (1-to-2 sentences max) explain how the above code results in a race condition

   variable num is read and modified by multiple threads.

3. Describe a critical section using a suitable C++ code fragment (don't write a whole program; but no code, no points. Most likely exam question) [**2 points**]

```
queueMutex.lock();
if(queue.size()<MaxQSize) {
queue.push(rand()%10000);i++;}
 else{idle++;}
queueMutex.unlock();
```

4. Four conditions must be met to establish an effective critical section in a multi-threaded program. Assume the adjacent `threadMain` method is called from many threads. State each condition. Then briefly (1-or-2 sentences) if-and-how the adjacent method meets or violates each condition. [**4 points**]

```
std::mutex mutex;
int shared = 0;
void threadMain() {
  std::lock_guard guard(mutex);
  shared++;
  // Sleep for few seconds
  sleep(10000 + rand());
}
```

i. No 2 threads in same CS simultaneously

ii. No assumptions about speed or number of cores/CPUs

iii. No thread outside a CS may block a thread in the CS

iv. No thread should wait too long (max ~1 millisecond wait time) to enter its CS

5. What is a semaphore? What is the difference between a semaphore and a mutex? [**2 points**]

Semaphore is a shared counter managed by the OS which ensures counter change are thread safe. Threads check value semaphore before operating on shared data. The basic difference between semaphore and mutex is that semaphore is a signaling mechanism, that is, the process performs wait() and signal() operations to indicate whether they acquire or release resources, and the mutex is a locking mechanism. If a process wants to acquire a resource, it must acquire a lock on the mutex.

6. What is a monitor (or a condition variable)? What is the difference between a monitor and a mutex? [**1 points**]

Monitors are higher level concepts than Mutex which do need a Mutex to operate. The monitor is limited to the current application domain. Mutexes can be used to synchronize threads across processes. When used for inter process synchronization, mutexes are called named mutexes because they will be used in another application and cannot be shared through global or static variables.

7. What is the producer-consumer model? Give one example scenario where this model is applicable [**1 points**]

Producer-consumer describes two processes which are producers and consumers. They share a fixed-size common buffer that acts as a queue. And the sample scenario is the thread pool of Java. The thread pool can execute or enter a blocking queue based on the current execution in the pool.

Code：

```
Void producer(constintnum);
Void consumer(constintnum);

int main() {
std::thread prod(producer, 500);
std::thread con(consumer, 500);
prod.join();
con.join();
return 0;
}
```

8. What is the dining philosopher's model? Give one example scenario where this model is applicable. [**1 points**]

Periodically they get hungry need to eat – for example, update shared resource what need 2 chopsticks 2 resources to eat, use one adjacent chopstick at a time– If chopstick is not free philosopher puts other chopstick down

```
oidphilosopher(intid) {
unsigned intseed = 0;
while(true) {
states[id] = THINKING;sleep(seed);
states[id] = HUNGRY;
 std::lock(sticks[id], sticks[(id+ 1) % 5]);
states[id] = EATING;sleep(seed);
sticks[id].unlock();
sticks[(id+ 1) % 5].unlock();
```

9. The adjacent method is counting number of "TATA" motifs (or substrings) in a full human `dna` (a 6 GB long string, note that 6 GB is well outside the range of `int`). This method is rather slow. Multithread this method to run faster using '*k*' threads. You may add any number of helper methods and variables as needed. [**3 points**]

```
int countMotifs(const std::string& dna) {
  int count = 0;
  for (size_t i = 0; (i < dna.size()); i++) {
    if (dna.substr(i, 4) == "TATA") {
      count++;
    }
  }
  return count;
}
```

Int real count;

```
Std::mutex countlock;
Int count;
Int real count;
Size_t times = k;
int countMotifs(int s, int e, const std::string& dna) {
 int count = 0;
 for (size_t i = s; (i < e;i++) {
  if (dna.substr(i, 4) == "TATA") {
   count++;
  }
 }
Std::lockguard<std::mutex> guard(countLock);
Realcount += count;
 return realcount;
}
Int main(){
Size_t length = dna.size()/k;
Size_t remain = dna%k;
Vector<thread> thrlist;
For (int I =0,i<k,i+=length){
Size_t ex = length;
If(end==k-1 && remain>0){
Ex += remain;
}else {
Ex += 3;
}
Thrlist.push_back(thread(count,I,ex,dna)));
}return 0;
}
```