Problem set 3

Part 1, Theory

Problem 1

a) What is cache memory?

Cache memory is a random access memory that a computer can access more quickly than it can access regular RAM. The purpose of cache memory is to store program instructions that are frequently referenced by software during operation. Fast access to the these instructions increases the overall speed of the software program.

b) What is the difference between spatial and temporal locality?

Temporal locality refers to accessing a memory location in time. Spatial locality refers to accessing a memory location in space.

Cache memories consider temporal locality of memory accessing because if a memory location is at one point referenced, then it is likely that the same location will be referenced again in near future.

Cache memories consider the spatial locality of memory accessing because if a particular memory location is referenced at one point, then it is likely that neighbouring and generally nearby memory locations will be referenced in the near future.

c) What is cache coherence?

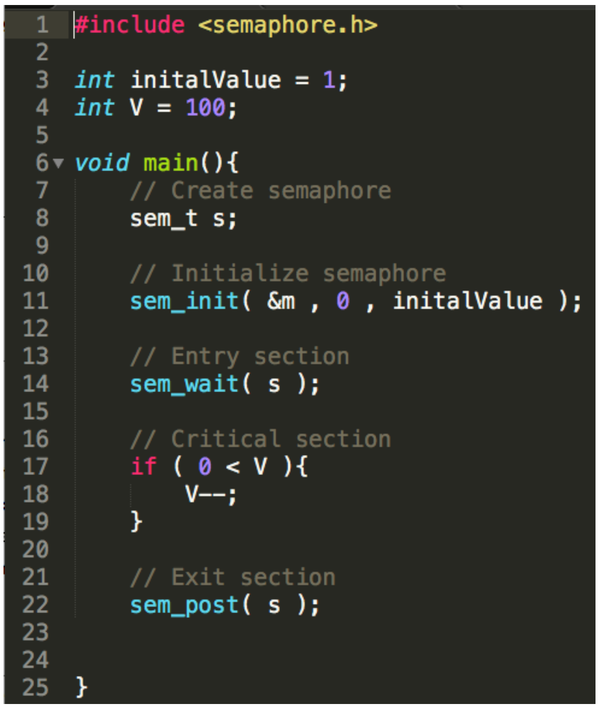
Cache coherence refers to the consistency of shared resource data that ends up stored in multiple local caches. In a multiprocessing system problems with inconsistent data may arise.

d) What is false sharing?

False sharing is an action which may arise in a multithread system. If multiple threads with separate caches access different variables that belong to the same cache line, updates performed by one of these threads may force the other threads to copy from main memory even do no changes have happened to the data belonging to that particular thread. This action is performance-degrading, but ensures cache coherence

Problem 2

a) Write code to show how semaphores can protect a critical section, when using multiple threads. Explain how your code works.



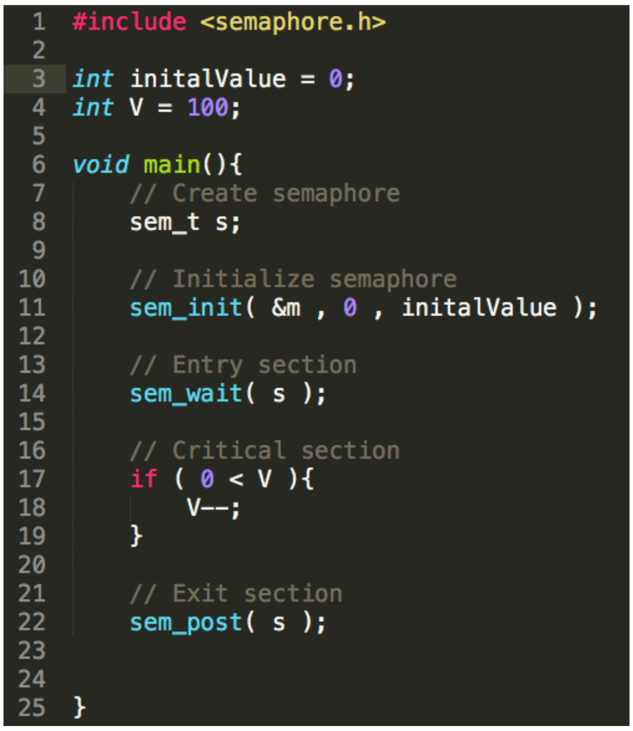
```c
#include <semaphore.h>

int initalValue = 1;
int V = 100;

void main(){
    // Create semaphore
    sem_t s;

    // Initialize semaphore
    sem_init( &m , 0 , initalValue );

    // Entry section
    sem_wait( s );

    // Critical section
    if ( 0 < V ){
        V--;
    }

    // Exit section
    sem_post( s );


}
```

The simple example above shows semaphores can be used to protect a critical section with a shared variable (here V). Using semaphores as locking variables we use the functions wait() and post() as lock and unlock of access to critical section. The function sem_wait() will decrement the value of the semaphore variable (here *s*) and access critical section if s after decrementing equals 0. The function sem_post will increment the value of the semaphore variable (here *s*) and exit the critical section.

For a multithread system thread 0 will decrement s and access the critical section. Thread 0 will hold access to the section until it increments s in sem_post. The behaviour of thread 1 (starting after thread 0) will depend on the status of thread 0. If thread 1 tries to access the critical section while thread 0 is inside, sem_wait() will decrement s to -1 and thread 1 will be blocked from entering critical section until thread 0 increments the variable.

---

b) Write code using semaphores that may deadlock when using multiple threads. Explain why your code may cause a deadlock.



```
 1  #include <semaphore.h>
 2
 3  int initalValue = 0;
 4  int V = 100;
 5
 6  void main(){
 7      // Create semaphore
 8      sem_t s;
 9
10      // Initialize semaphore
11      sem_init( &m , 0 , initalValue );
12
13      // Entry section
14      sem_wait( s );
15
16      // Critical section
17      if ( 0 < V ){
18          V--;
19      }
20
21      // Exit section
22      sem_post( s );
23
24
25  }
```

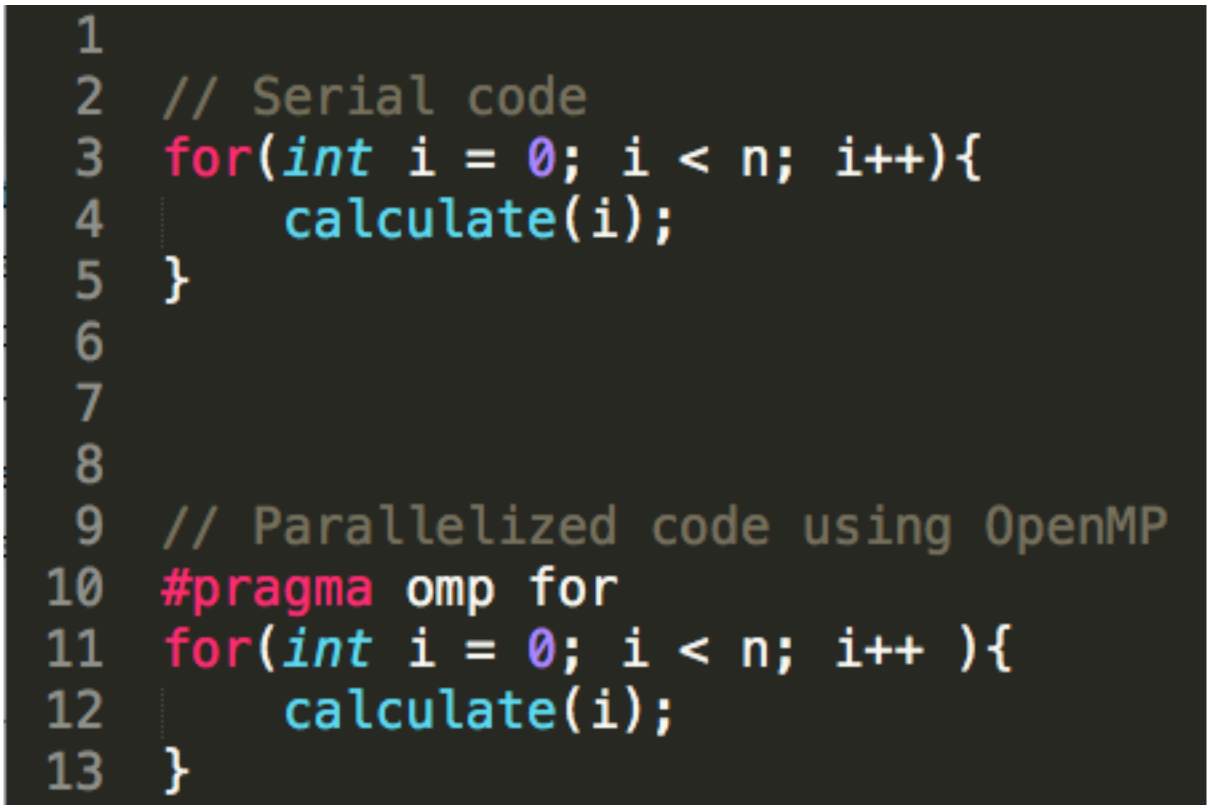
This the same example as above with one change at line 3. The initial value of the semaphore variable is now set to 0 instead of 1. This will lead to sem_wait decrementing the variable to -1 for first access and block access, and no thread will access the critical section. The code will leave all threads deadlocked.

Problem 3

a) What is the difference between OpenMP and pthreads?

Pthreads requires that the programmer explicitly specify the behaviour of each thread. OpenMP, on the other hand, sometimes allows the programmer to simply state that a block of code should be executed in parallel, and the precise determination of the tasks and which thread should execute them is left to the compiler and the run-time system. Pthread can be used with any C compiler, provided the system has a Pthreads library. OPenMP require compiler support for some operations. Pthreads is developed for usage at lower level programming, but OpenMP is developed to be used at higher level programming.

b) Show how the following code can be parallelized using OpenMP



```

// Serial code
for(int i = 0; i < n; i++){
    calculate(i);
}



// Parallelized code using OpenMP
#pragma omp for
for(int i = 0; i < n; i++ ){
    calculate(i);
}
```