Spiro Mavroidakos -

Jastaj Virdee - 260689027

ECSE 420 - Assignment 1

# Q1

1. l
2. The tasks that were defined in the code that work in parallel are matrix multiplication algorithms that only produce one row of the product matrix. In other words if matrix A is multiplied with matrix B to produce matrix C, each of the tasks will produce one row of C. Hence after all the tasks are completed, the entire product matrix will be produced.
3. K
4. As seen in the graph in 1.4, the execution time decreases after the first two increments of the number of threads, however it then levels off. The reason for this is that the CPU being used only has two cores, so adding more than two threads will not aid in decreasing the execution time, which is why the time levels out.

# Q2

1. Deadlock can occur when 2 or more threads need to acquire the locks on several shared objects. Deadlock happens when each thread holds a lock that the other needs and therefore leads to all the threads waiting for a lock that that they cannot get.

There are 4 conditions for deadlock to occur:

1. Mutual exclusion
2. Hold and wait: A thread is holding a resource while waiting to get another.
3. No pre-emption: A thread will not give up a resource until it finishes using it.
4. Circular wait: Each thread holds a resource that another needs.

Conditions 1,2 and 3 are necessary but not enough for deadlock to occur. All 4 conditions are needed.

1. One solution would be resource ordering. In this solution, an order is assigned to each object whose lock needs to be acquired and ensure that each thread acquires the locks in that order.

Another solution would be to evaluate if granting the lock will cause the system to deadlock. For this to happen, every thread would need to send a list of all the shared resources that it needs. The system will also keep track of how much of a specific resource is available. The system will then receive requests for resources and deem if granting this request will lead to deadlock. If found that the request will make it so that no other thread can terminate due to being inability of getting a resource, then the request gets denied and received more requests, granting only those that are safe. This is known as Banker’s algorithm.

# Q3

1. A
2. To avoid deadlock, the solution is simple. Create one lock (re-entrant lock) to act as a moderator. When a philosopher gets hungry, they will ask the moderator if they can acquire the chopsticks (acquire the lock before the locks on the chopsticks). If the philosopher can acquire the lock then they have access to the chopsticks. The only issue with this approach is that only one philosopher will be able to eat at a time but that also means that deadlock will not occur.
3. Using the solution for 3.2, we can simply set the reenetrant lock to implement fairness by setting it (i.e. lock = new ReentranLock(true)). As seen in class, fair locks prevent starvations as this ensures that every thread has fair access to the shared resources. Fairness implies that every thread will be able to access the resources they need.

# Q4