Q1(a).

A distributed system is a system which includes multiple different networked computers where their components are located. These computers pass messages between each other to communicate and coordinate their actions. Parallel system is one that can support multiple events occurring at the same time. An example of a distributed system is a network provided by a phone company while an example of a parallel system would be an operating system which has many processes occurring at the same time. For this task, I believe a parallel system would be more suitable as coarse-grained parallelism involves multiple loops running at the same time, so a parallel system is better suited to handle this and would lower communication costs. I think a distributed system would better to optimize throughput as it can use multiple machines on its network to process task while a parallel cannot do this.

(b)

Core 4

Core 3

Core 2 Core

Core 1 Core

Reports

Network Comms

User Interface

Sensor

Scheduler

(c)

S = 1/B+1/n(1-B)

= 1/0.1+1/100(1-0.1)

= 9.17

Q2

(a).

Safety properties are properties that must always be true while liveness properties are properties that must eventually be true. Examples of safety properties are mutual exclusion to ensure that two processes never interleave certain sequences of instructions and absence of deadlock. Examples of liveness is the absence of starvation and ensure there is fairness. Locks will help with liveness properties as they provide mutual exclusion so that property will always be true while locks are in use and can also use lock methods to ensure a lock is not already in use which helps prevent deadlock. For liveness properties, having short critical sections will help reduce the chances of starvation also ensuring each thread get the right time they need will help ensure fairness. In my assignment work I used these properties to ensure that the program was not being affected by deadlock, at the start we were facing the issue where the code if we ran over a certain number of events at once then the program would just hang and not finish however at this point, we had not implemented locks, so this was clearly the issue. We then implemented locks to check if a lock was already in use and if it was the thread waited for it to be released.

(b).

i)

The critical section is from line 6-9

ii)

the Answer should be protected because its private however veryBig is not as it is defined inside the calcAnswer function and left public.

III)No because if there are two threads looking to use the answer at once then this will cause deadlock so locks must be added to check if thread is currently using the answer variable to ensure this is avoided.

Iv)

private boolean isReady = false;

private BigInteger theAnswer = 0;

private static final int aBoundary = 10000;

// calcAnswer runs in first thread

private void calcAnswer() {

acquire(myAnswerLock);

BigInteger veryBig = new BigInteger(aBoundary, new

Random());

// busy for a long time ...

theAnswer = veryBig.nextProbablePrime();

release(myAnswerLock);

isReady = true;

}

// useTheAnswer runs in second thread

private void useTheAnswer() {

while (!isReady) {

Thread.yield();

}

if (theAnswer == 0) throw new RuntimeException("The

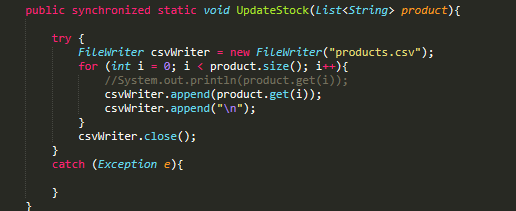
answer was not ready!");

}

(c). To avoid deadlock at least one of the 4 requirements must not be met. In this case adding locks will ensure that no resources will break priority which is one of the requirements of deadlock, this is because every time the resources are to be used by a thread, it will check to see if the lock has been held by another thread and if it has it must wait until it is released. This also ensures that the hold-and-wait requirement is also not meet.

(d).

In the 2nd assignment where we used Java RMI to develop a product ordering system we had to ensure that it no user could access products at the same time so we had to use locks to stop them from to access them. We were using Java due to using Java RMI so we were able to use the synchronized keyword implement locks and to solve this



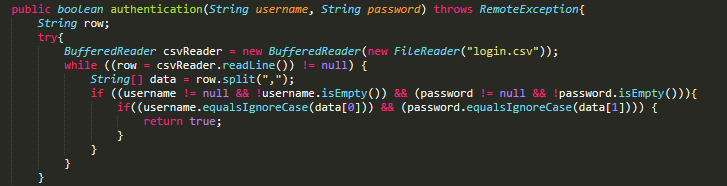
Q4

(a)

For project work we had to use web services to develop a client-server application implementing an online ordering system. This system had to include multiple features such as running 2 distinct machines at once connected by a network, include a login, and allow customer to do many different commands such as make an order and check current and future product stock. The main design decision we made was to use Java RMI, this decision was mainly based on time we had and being unfamiliar with other approaches so we felt this would be the best way to get a high-quality system produced.

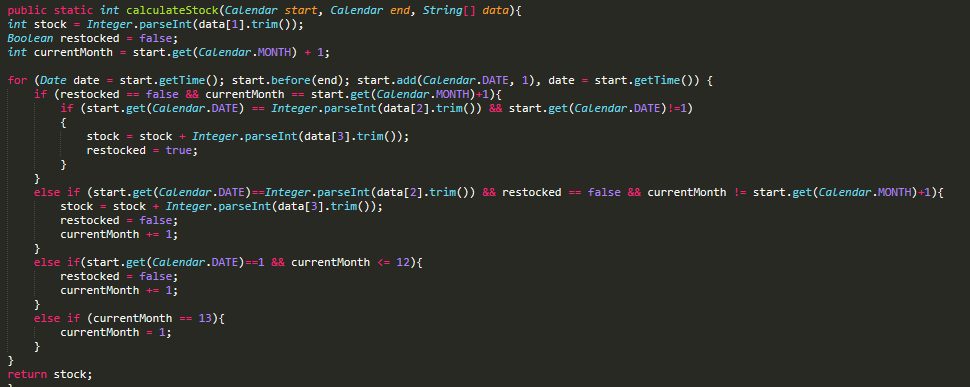
The next decision we had to make was how we would store the product information and login details. We initially investigated JSON files as we had been using them in another project, but this seemed to not as straight forward as other options that could be implemented easier and offer the same functionality, so we ended up using csv files.

We decided to get basic functionality working at first before implementing things such as threads and locks, so we started with the basic features such as login which was just checking from a csv file whether the login details entered existed in csv file.



Other features were checking stock and making an order which were basically just writing to and reading from the csv files at this stage. Then when we had some basic functions working, we implemented the threads using executors which was used due to it allowing threads to run concurrently and we were very familiar having just used them in the previous assignment.

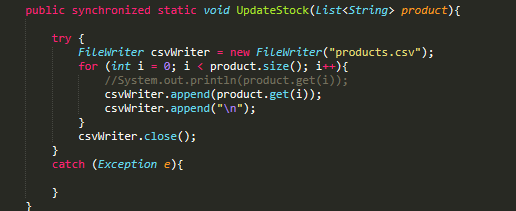
We then continued to add more advanced features such as updating stock when orders were made and checking future stock. Checking future stock was quite difficult to get calculations correct as to do this we has to use the Calendar library and set a start time and the end time which the user enters and iterate day by day until the end date was reached.



This involves having to implement several checks to ensure that the stock wasn’t being restocked multiple times a month and that if the end of the year was reached that it would return to January and not month 13 which would cause calculation to not take account of any months after this. I believe this implementation could have been improved on however we did not have time to go back and make this calculation more efficient as going day by day is not ideal and for long periods may take a while to run.

Finally, after all the main functions were working as we expected we implemented that ordered could be cancelled which was simply just changing the status on csv file to “cancelled” and adding order total back to the current stock of the product ordered. Rewriting the files was also another area which could have been improved as each time the file was edited, we had to completely rewrite the file which we could not find a way around this and are still unsure whether this is possible using csv files.

After completing all the features, we implemented locks which we decided to use the synchronized keyword as we were using Java, this made implementing locks quite straightforward for us and found little issues with deadlock, starvation after implementing them.



Synchronized was only added to functions that involved changing a resource such as the update stock method shown above.

(b).

Our system would support the concept of location transparency as the user would not be aware of the location of objects in the system as it is being used from the command line and all the user see is the responses returned to them each time they enter a command.