INDIVIDUAL ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT074-3-2

CONCURRENT PROGRAMMING

APD2F2109CS(DA) / APU2F2109SE / APU2F2109CS(DA) / APD2F2109CS / APD2F2109SE / APU2F2109CS

HAND OUT DATE: 11 MARCH 2022

HAND IN DATE: 22 APRIL 2022

WEIGHTAGE: 20%

NAME: David Wee Ming Junn

TP: TP062496

INSTRUCTIONS TO CANDIDATES:

1. **Submit your assignment at the administrative counter.**
2. **Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing).**
3. **Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld.**
4. **Cases of plagiarism will be penalized.**
5. **The assignment should be bound in an appropriate style (comb bound or stapled).**
6. **Where the assignment should be submitted in both hardcopy and softcopy, the softcopy of the written assignment and source code (where appropriate) should be on a CD in an envelope / CD cover and attached to the hardcopy.**
7. **You must obtain 50% overall to pass this module.**
8. **This report is Part 1 of 2 for the assignment.**

Abstract

The goal of this report is to propose a proposal for the structure of the simulation to evaluate the performance of the airport's system. Most of the assumptions for the program are reflects the requirements given. Next, the flow of activities has been illustrated from requesting landing till take-off while including concurrency concepts to be applied and seconds taken for each action to conclude. Moreover, the concurrency techniques to be implemented are further justified with examples. Lastly, expected conundrums are envisioned and safety measures are outlined to minimize impact of a blunder towards the simulation's development.

Contents

Abstract 2

1.0 Introduction 3

2.0 Requirements 3

3.0 Assumptions 4

3.1 Seconds Required for Actions 4

4.0 Flow of activities 5

5.0 Concurrency Concepts 6

5.1 Synchronization 6

5.2 Semaphore 6

5.3 Wait and Notify 6

6.0 Safety Aspects of Multi-Threading 7

6.1 Interleaving and Data Race 7

6.2 Deadlocks 7

Conclusion 8

References 8

1.0 Introduction

The management of a small airport speculates that the current system utilized by the airport are by no means efficient. Incoming flights are suspected to have a prolonged time of waiting for landing clearance. A simulation prototype is expected to be created to evaluate the performance of the system by attempting to imitate description of real-time events occurrence in the airport while collect small amounts of statistical data. In other words, the purpose of this report is to create a concurrent program that imitates the function of an Air Traffic Controller(ATC) simulation that does not require real-time events yet. This report serves to be a foundation for the forthcoming simulation though multiple error detection and avoidance in programming. Hence the purpose of this report is to propose the structure of the simulation.

2.0 Requirements

These are the requirements given by the case study.

1. There is only 1 runway for all planes to land and depart.

2. Aircrafts must not collide on the runway or gates.

3. Once an aircraft obtains permission to land, it should land on the runway, coast to the assigned gate, dock to the gate, allow passengers to disembark, refill supplies and fuel, receive new passengers, undock, coast to the assigned runway and take-off.

4. Each step should take some time.

5. A congested scenario should be simulated where planes are waiting to land while the 2 gates are occupied.

6. As the airport is small, there is no waiting area on the ground for the planes to wait for a gate to become available.

7. Passengers disembarking/embarking, supply refills and aircraft cleaning should happen concurrently.

8. There is only one refuelling truck.

3.0 Assumptions

1. Based on requirements 1 and 2.
    - Only one plane may be on the runway at any given time.

2. Based on requirement 4.
    - Each action will take several seconds to complete.

3. Based on requirement 5 and 6.
    - There are only two gates at the airport.
    - Gates' availability defines the availability of a plane to land.
    - Only two planes may be on ground at a time.
    - Gates will not be considered available if its respective plane have not taken-off.

4. Based on requirement 7.
    - Passenger disembarking/embarking, supply refills and aircraft cleaning cannot happen simultaneously.

5. Based on requirement 8.
    - Refuelling truck can only be used by one plane at a time.

6. Only one plane can use the runway at any given time.
    - Since the airport is small, it is unlikely that the runway is big enough for a plane to enter the runway and wait while another plane lands.
    - Safety of passengers is compromised when two planes use the same runway to land and depart at the same time.

7. Only 6 planes will land for this simulation.
    - This is just a prototype hence it is just to show that the program is working as intended.

8. The refuelling truck has enough fuel for 6 planes.
    - The refuelling truck does not need to refuel itself during the program.

3.1 Seconds Required for Each Actions

9. Plane landing takes 4 seconds.
10. Plane take-off takes 4 seconds.
11. Planes will arrive randomly between 0-3 seconds.
12. Planes coast and dock to the gate takes total of 1 second.
13. Fuel refilling takes 1 second.
14. Aircraft cleaning takes 1 second.
15. Passenger embarking and disembark takes 1 second each.
16. Supply refill takes 0.5 seconds.
17. Undock from the gate takes 0.5 seconds.
18. Coast from the gate to runway takes 0.5 seconds.

4.0 Flow of activities

(n). n = numbers. To represent the number of seconds required by the action.

Exp: run(n) = run takes n seconds.

- When the simulation begins, planes will start to arrive and request for landing. The planes will be put on queue until the runway and a gate is available which will be notified by the ATC. LinkedList will be used in this scenario instead of ArrayList. The reason for LinkedList is because it is faster than ArrayList in the terms of addition and deletion of the queue as no reading is required. Lastly, The queue will naturally be First in First Out to ensure equity.

- The availability of the gates will be monitored by the ATC through semaphore object. ATC will not provide the green light to land when the gates are unavailable. Lock is acquired individually when a gate is occupied.

- When the runway is clear and there's a gate available, the ATC will notify the first plane in the queue to land along with the gate label. Planes will acquire the runway using synchronized while it lands on the runway using 4 seconds. The semaphore object monitoring the number of available gates will immediately acquire a lock prior. The runway will not be accessible by other planes during the plane's landing as it will be locked and unlocked after landing has ended. This is a safety measure to ensure 0 collision.

- The planes will then head(0.25) and dock(0.25) at the gate previously notified, using 1 second. Then, passengers will disembark, supplies and fuel(1) will be refilled, aircraft will be cleaned, new passengers will embark. Passengers disembark and fuel refill may happen simultaneously. Supply refills(0.5) and aircraft cleaning(1) will only happen after passengers have disembark(1). Passengers will then embark(1) after all the operations are done. This sequence may easily be configured through wait() and notify(). Moreover, synchronize will also be used to ensure the refuelling truck can only be used by a plane one at a time.

- After that, the planes will undock(0.25) and request for take-off. If the runway is clear, the plane will be notified and proceed to the runway(0.25) and take-off(4). As previously mentioned, the runway can only accommodate one plane at a time, hence if a plane is landing, take-off will only occur subsequently. The semaphore will release the lock.

- These activities will repeat until 6 airplanes has landed and departed. Subsequently, the simulation will cease.

5.0 Concurrency Concepts

Straightforwardly, concurrency concepts refer to the concept of two or more tasks execute parallel to each other. As an example, two or more threads with several identical actions running at the same time without producing an error.

5.1 Synchronization

The synchronized keyword in Java provides locking, which ensures mutually exclusive access to the shared resources and prevents data race. (Oracle Corporation, 2010) As witnessed in the flow of activities, when a plane is allowed to take-off or land on the runway, the plane will lock the runway. The runway can only exclusively be used by the plane until it finishes its action. When the action is finished, the lock will then be released through synchronize.

Furthermore, synchronized is also used for the refuelling truck to ensure that a refuelling truck is not used by more than one planes at a time.

5.2 Semaphore

Semaphore controls access to a shared resources through the use of a counter. If the counter is greater than zero, then access is allowed. Else, access is denied. (GeeksforGeeks, 2018) A semaphore object is expected to be implemented in the program. Starting with a value of 2, when a gate is acquired, the value will decrement. When value is 0 and attempting to decrement, it will not. Action is forced to wait until value is incremented or a gate becomes free.

5.3 Wait and Notify

Wait and notify are important in concurrency. They are to maintain coordination between different processes through condition and confirmation. When a thread is waiting for an action to finish, it will forever stop until it is notified. There are multiple situations in which several actions need to stop before continuing such as when the plane has dock on a gate. There are several actions that must be done. Some can be done simultaneously, while some can only be done after other actions are done. Hence the importance of wait and notify.

6.0 Safety Aspects of Multi-Threading

What requires a safety net when programming multi-threading? The answer is critical sections. A critical section is a block of code that accesses a shared resource and can't be executed by more than one thread at the same time. (Patrawala, 2018) As demonstrated in the flow of activities, the runway is a shared resource that belongs to a critical section.

6.1 Interleaving and Data Race

Interference happens when two operations, running in different threads, but acting on the same data, interleave. This means that the two operations consist of multiple steps, and the sequences of steps overlap. (docs.oracle) When interleaving occurs, data race occurs. In summary, data race refers to the act of threads receiving false expected value due to the manipulation and reading of multiple threads. Before the shared data is able to be updated, it has been read by another thread, hence causing errors.

To prevent this from happening, the functions provided below concurrency concepts is used to ensure that all critical sections are given a safety net to ensure that data race does not occur. For example, the fuelling truck has been instructed to fuel plane A but before the fuelling truck can be listed as 'occupied', it is suddenly instructed again to fuel plane B. Hence the occurrence of one fuelling truck refuelling two planes which is not at all possible. The use of synchronize ensures that when a fuelling truck is instructed to fuel a plane, it is locked. The fuelling truck cannot be instructed to go elsewhere until the lock is released.

6.2 Deadlocks

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. (tutorialspoint) To avoid this, the program will use as less unnecessary locks as possible. Locks will only be used for critical sections such as the runway and the gates availability. Moreover, nested lock will be prevented as much as possible. For reference, after the plane has landed, the lock on the runway will immediately be released, instead of holding the lock on the runway and proceed to acquire a lock on the gates.

Conclusion

The requirements provided by the airport regarding their ATC has been fulfilled and converged with the assumptions of this program. Next, the flow of activities of the program has been discussed greatly in detail including the concurrency concepts and functions which will be coded within the program. Moreover, expected errors has been defined and safety measures has been indicated as well to ensure smooth execution. Overall, this proposal is ready to be implemented.

References

- docs.oracle. (n.d.). *Thread Interference.* Retrieved from https://docs.oracle.com/javase/tutorial/essential/concurrency/interfere.html
- GeeksforGeeks. (10 December, 2018). *Semaphore in Java.* Retrieved from https://www.geeksforgeeks.org/semaphore-in-java/
- javaee.github.io. (n.d.). *Concurrency Basics.* Retrieved from https://javaee.github.io/tutorial/concurrency-utilities001.html#:~:text=Concurrency%20is%20the%20concept%20of,single%20CPU%20is%20very%20common.
- javatpoint. (n.d.). *How to Avoid Deadlock in Java.* Retrieved from https://www.javatpoint.com/how-to-avoid-deadlock-in-java
- netjstech. (2 September, 2021). *Race Condition in Java Multi-Threading.* Retrieved from Tech Tutorials: https://www.netjstech.com/2015/06/race-condition-in-java-multi-threading.html
- Oracle Corporation. (2010). *What is a Data Race?* Retrieved from https://docs.oracle.com/cd/E19205-01/820-0619/geojs/index.html#:~:text=A%20data%20race%20occurs%20when,their%20accesses%20to%20that%20memory.
- Patrawala, F. (30 May, 2018). *Java Multithreading: How to synchronize threads to implement critical sections and avoid race conditions.* Retrieved from https://hub.packtpub.com/java-multithreading-synchronize-threads-implement-critical-sections/#:~:text=A%20critical%20section%20is%20a,programming%20languages)%20offers%20synchronization%20mechanisms.
- Richard, P. (6 February, 2018). *When to use LinkedList over ArrayList in Java.* Retrieved from https://www.tutorialspoint.com/When-to-use-LinkedList-over-ArrayList-in-Java#:~:text=LinkedList%20should%20be%20used%20where,rarely%20modified%2C%20ArrayList%20is%20suitable.
- tutorialspoint. (n.d.). *Java - Thread Deadlock.* Retrieved from https://www.tutorialspoint.com/java/java_thread_deadlock.htm