|  |
| --- |
| NAME : VIVEK BANSAL STUDENT ID : 112044493 MAIL ID : VIVEK.BANSAL@STONYBROOK.EDU COURSE NUMBER : CSE 535 (ASYNCHRONOUS SYSTEMS) ASSIGNMENT NUMBER : 1 DUE DATE : 03 SEPTEMBER 2018 |

# ANS 1:

* Attached is the PDF file (week1\_1.pdf) with my comments mentioned there. I have added comments using Adobe Acrobat Reader.

# ANS 2:

* **Problem:** User wants to book a cab from an app called Uber. Once he books a cab from his/her app, a cab should come at his location in the minimum possible amount of time as well as with less fare for the ride.
* **Input:**
* User’s current location coordinates from his GPS.
* Which type of cab user wants to book – UberX or UberXL.
* User’s destination location coordinates.
* **Output:**
* A cab should come at user’s location in minimum possible time as well as with less fare for the ride.
* **Interesting Fact about this problem:**
* This problem is interesting to me as it is a real-world problem and it impacts many people’s lives and ease their task of moving from one place to another. I am fascinated about how efficient the application is in booking a cab in very less time by just some clicks of the buttons.
* **Distributed Algorithm Use:**
* Suppose the user wants to book a cab from Stony Brook University in New York State. The query is send to the centralized server of Uber app which will divide the query into multiple sub-queries and send them to the different systems of distributed architecture. The different machines contain data about the cabs available in different states. For example: 3 machines may contain all cabs availability data in New York, 4 machines may contain all cabs data in California etc. In our case, queries will be send to 3 different machines containing data about New York. These machines will check the cabs available in Stony Brook University of New York and return their respective results to the centralized server. Now server will check which cabs are at least distance from user’s current location. It’s a trade-off between distance and fare and there must be some other mechanisms to resolve this. So as we can see this problem requires distributed algorithm to efficiently break down user query into many sub-queries and giving them to different systems and finally combining their results to give a final solution.

# ANS 3:

* **Problem Statement:** Implement message passing using shared memory. I am trying to explain synchronous communication (blocking calls) mechanism. Any Process can read and write data into the buffer. The system is scalable to any number of processes. And each process knows its destination recipient like which process it needs to send its data.
* **Input:**
* 2 Global variables:

Lock = FALSE

isDataModified = FALSE

* A buffer to hold data which acts as the shared memory.
* **Solution:**
  + - * **Send Event:**

If the lock == TRUE, it will wait for the lock to be released.

Once lock == FALSE, it will set it to TRUE to achieve mutual exclusion and then it writes the data into the buffer. After writing data it will set isDataModified = TRUE to signal that data is modified and set lock = FALSE to release the lock.

* + - * **Receive Event:**

Keep checking the isDataModified == TRUE in a loop.

Once isDataModified is set this means that data is modified and this process wants to read it.

Now it will check the lock. If lock == TRUE, this means other process is already into the critical section so this process has to wait until the lock will be released.

Once lock == FALSE this process will set it to TRUE and read the data from the buffer. After reading the data from buffer it will set isDataModified == FALSE to signal data read and lock == FALSE to release the lock.

* **Pseudo Code:**

|  |  |
| --- | --- |
| **SEND EVENT** | **RECEIVE EVENT** |
| if (lock || isDataModified))  Wait();  lock = 1;  Write (msg, dest);  isDataModified = 1;  lock = 0; | if (!isDataModified)  Wait();  If(lock)  Wait();  lock = 1;  Read ();  isDataModified = 0;  lock = 0; |

# ANS 4:

* We are taking 2 possible scenarios in this case:

1. **First Scenario:**

* Input: Let’s say we have 2 processes Pi and Pj with logical clocks as Ci and Cj respectively.
* Solution: Process Pi sends a message with timestamp Tm to Pj. After receiving the message from Pi, Pj will set its logical clock Cj = max (Cj, Tm) + 1. This is the maximum possible value of lamport clock at this time.

1. **Second Scenario:**

* Input: Let’s say we have n processes P1, P2, P3, … Pn each with logical clocks as C1, C2, C3, … Cn.
* Solution: The maximum value of lamport clock of the system at this point of time is: max (C1, C2, C3, … Cn).

# ANS 5:

* **Input:** Let’s say Process Pj sends the message with timestamp Tm to process Pi.
* **Solution:** After receiving the message from process Pj, Pi should set Ci to be greater than its present value of its logical clock (and not equal to its present value) and greater than Tm. This is because if present value of logical clock at process Pi is greater than Tm, process Pi may set its logical clock equal to its present value (as equality sign is there in the expression). This is not correct as this means that 2 events at Pi (this receiving event and preceding event) happen at the same time as they have the same timestamp which is not a possible scenario. So equality sign should be removed from the expression and it should be strictly greater than sign.