Tutorial 5 (CSN-502)

Submission due by 5 PM 16th Oct 2020

1. Write a program in C++ / Java to implement Lamport’s logical clocks. Your program should take as input a description of several process schedules (i.e., lists of send, receive or print operations). The output of your program will be a linearization of these events in the order actually performed, annotated with Lamport clock values.

The input of the program will be a collection of processes, each with a list of operations to perform. The processes are named p1...pn for some n (you may assume that n is at most 9) The format of a process is:

begin process p1

operation

…

operation

end process

where each line contains a basic operation. The possible basic operations are:

● send pN msg (that is, send message msg to process pN)

● recv pN msg (that is, receive message msg from process pN)

● print msg (that is, print message msg to the terminal)

where msg is any alphanumeric string.

The send operation simply sends a message and the process is free to continue executing the next operation. The recv operation blocks and waits to hear message msg from a given process. (This means that there can be deadlocks if all processes are waiting to receive and there are no messages in transit.). An individual print operation takes place atomically.

Messages can be sent and received, and printing can take place, in any order, provided causality is respected: that is, the order of events within a process is preserved, and a message is always sent before it is received.

One approach to handle send and recv operations is to maintain a pool of messages including sender, receiver and payload. When a message msg is sent from p1 to p2, we add message m = (p1, msg, p2) to the pool, and when the message is received by p2, m is removed from the pool.

Here is a small example illustrating the input format:

begin process p1

send p2 m1

print abc

print def

end process

begin process p2

print x1

recv p1 m1

print x2

send p1 m2

print x3

end process p2

Note that the message sent from p2 to p1 is never received, this is fine.

The output format is a single log of the events that took place during the simulation run, one per

line, including Lamport clock timestamps. The possible events are:

● sent pN msg pM T (that is, pN sent message msg to pM at local time T)

● received pN msg pM T (that is, pN received message msg from pM at local time T)

● printed pN msg T (that is, pN printed message msg to the shared terminal at time T)

The following is a valid output:

printed p2 x1 1

sent p1 m1 p2 1

received p2 m1 p1 2

printed p1 abc 2

printed p1 def 3

printed p2 x2 3

sent p2 m2 p1 4

printed p2 x3 5

Several other outputs are also possible depending on the order in which events at different processes happen.

Your program should report if a deadlock happens. For example if after the occurrence 5 events all processes are deadlock the output show those 5 events along with their clocks and then should print “system deadlocked”.

1. Repeat the above exercise for vector clocks.

Submit the following:

1. Your program
2. A README file stating that how your program should be complied and executed
3. Sample input (on which you have tested your program)
4. Output for the above input