Tutorial 6 (CSN-502)

(Submission date 9-10-2023 by 11:59 PM)
(Checking on 10-10-2023. Time will be informed later)

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's 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 10) 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 (pj, …, pk) msg (that is, send message msg to all processes in the list (pj, …, pk))
- 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 pi to (pj,…., pk) we add messages (pi, msg, pj), …, (pi, msg, pk) to the pool, and when the message is received by the process pj, message (pi, msg, pj) 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's clock timestamps. The possible events are:

● sent pN msg (pj, …, pk) T (that is, pN sent message msg to (pj, …, pk) 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 of n events all processes are in deadlock, the output should show those n events along with their clocks and then should print "system deadlocked".

Submit the following:

1. Your program
2. A README file showing
   (i) how your program should be complied and executed
   (ii) Sample input (on which you have tested your program)
   (iii) Output for the sample input

Submit a zip / rar file (name your enrollment no) containing your program and README file

Marks: Your program runs on your input – 35
Your program runs on our correct input (different from above) – 35
Your program detects our incorrect input and gives error message – 10
Along with error message your program also indicates the type of error – 10
Your program can handle deadlock – 10