Name- Ishu Gupta

Enrollment Number- 20114041

Email id- ishu_g@cs.iitr.ac.in

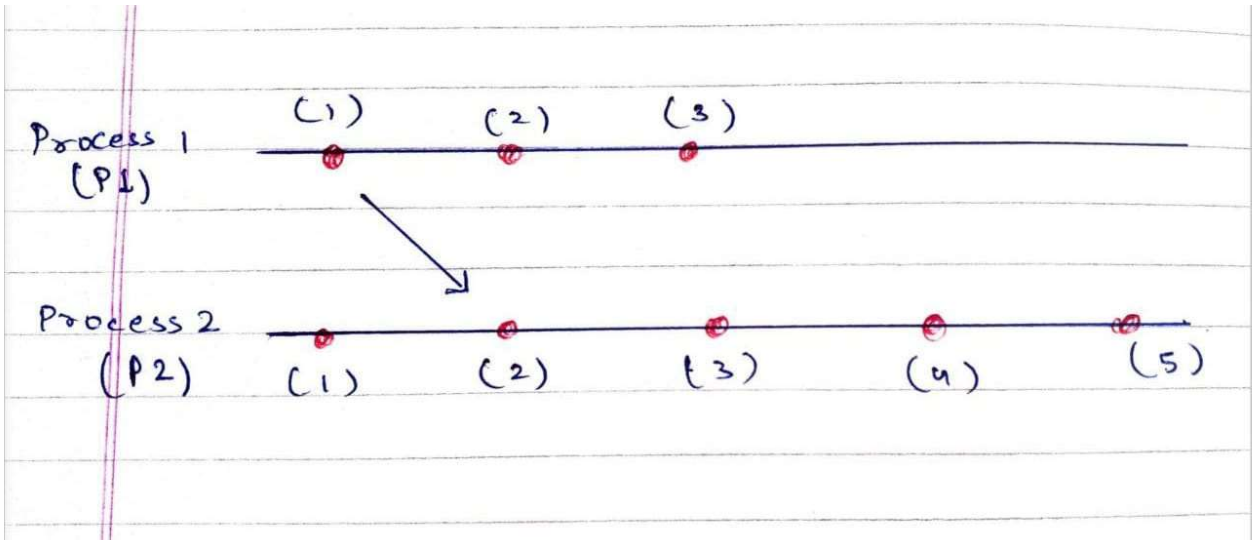
Phone number- 9001723540

Rule for updating timestamps-

If any “send” event is there, then for its corresponding “recv” event, timestamps will be updated according to the following rule-

```
If timestamp of “send” > timestamp of “recv”, then
    Step 1: difference= timestamp of “recv” - timestamp of “send” +1
    Step 2: Then, add difference to “recv” event and all other events after it in
    the same process.
otherwise, don’t change anything.
```

Diagram for the Example given in instructions:



Explanation of the code:

<u>Taking Input</u>:

I have taken input in LinkedHashMap with String as key (for storing process name like p1, p2, etc) and value as an ArrayList of String for storing all events of a given process.

Reason: I used Map because I wanted to store key- value pair since it is not necessary that user will give input in order of process number (i.e., first p1, then p2, and so on). So, to take random order of processes into consideration, I used Map.

<u>Finding Timestamps and printing output</u>:

I first assigned timestamps to all the events in the order in which they appear in a process and then iterated over all the ArrayLists multiple times to change timestamps as follows-

I did following in a while(true) loop-

- For every iteration of while loop, I iterated on the LinkedHashMap-

  Starting from the first value in the LinkedHashMap (ArrayList corresponding to the first process), if any “send” event is found, then search for the corresponding “recv” event, if it is found, then update values of timestamps according to the following rule-

    If timestamp of “send” > timestamp of “recv”, then

    1. difference= timestamp of “recv” - timestamp of “send” +1
    2. Then, add difference to “recv” event and all other events after it in the same process.

    otherwise, don’t change anything.

- Break condition of while(true) loop- If no timestamp is changed in some iteration of while loop, then, we have got final timestamps of all events listed in input

  But

  In case of deadlock, this loop could run infinite times, so here, I have taken one assumption-

  If loop runs more than 100 times, then the system is deadlocked.

Now, I got timestamps of all events listed in input.

Now, I printed all the events in order of their timestamps, by iterating from timestamp=0 to till a value when it's not found (by observation, I found that if an event with timestamp x+1 exists then event with timestamp x will definitely exist).

Deadlocks:

To take deadlocks into consideration, if any value of timestamp is not found but timestamp greater than this value exists for any of the events, this will be a condition of deadlock.

For this, I checked if there exists any timestamp greater than the one which is not found, if yes, then deadlock, if no, then output is printed successfully.

But, this condition is already taken into consideration earlier(number of iterations>100).

Assumptions:

1. For any "recv" event, there exists a corresponding "send" event.

To run the program, please ensure the following-

1. After the end of the last process, press enter to print the output.
2. Don't keep extra spaces in the input because I have used substring method to parse the events, so extra spaces between or after the line will give wrong output.

Sample inputs

- **These test cases are the sample inputs given in instructions' doc, I have just removed the extra spaces, etc according to my program.**

Test case 1:

```
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
```

Test case 2:

```
begin process p1
recv p4 m4
send p2 m1
end process
begin process p2
recv p1 m1
send p3 m2
end process p2
begin process p3
recv p2 m2
send p4 m3
end process
begin process p4
recv p3 m3
send p1 m4
end process
```

Test case 3:

```
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
```

Test case 4:

```
begin process p1
send p2 m1
send p2 m2
send p2 m3
send p2 m4
print m5
recv p2 m11
print m6
send p3 m7
end process
begin process p2
recv p1 m3
recv p1 m1
recv p1 m2
print m8
print m9
send p3 m10
send p1 m11
send p3 m12
end process
begin process p3
print m13
print m14
recv p1 m7
recv p2 m10
print m15
recv p2 m12
end process p3
```