# CS 171 Spring 2020
# Programming Assignment I

Assigned: April 4th, 2020
Due On Demo Day: April 17th 2020

**PLEASE NOTE:**

- Assignment Goal: understand Lamport clocks and socket programming.
- Students may work in pairs (2 people max) for this assignment.

## 1 Introduction

Lamport logical clocks are used to order events capturing causality or happens before relationship between events. An event e happens before another event $f$ (denoted by $e \rightarrow f$) iff:

- the same process executes $e$ and $f$, and e is executed before f;
- $e$ is the send event of a message $m$ and $f$ is the corresponding receive of a message $m$ (i.e., $e = \text{send}(m)$) and $f = \text{receive}(m)$); or
- $\exists h \mid e \rightarrow h \wedge h \rightarrow f$

In this assignment, you are required to write a program that assigns Lamport clocks to events happening on several processes. These processes communicate with each other using sockets. There is no shared memory between the running processes. A network process is used to simulate delays in the network.

## 2 Project Overview

In this project, you will be implementing 3 different processes that will communicate with each other, plus a network process. Each process will need to maintain a Lamport clock for it's local events, as well as communication events from the other 2 processes. All communication should be routed through a network process, which is used to simulate delays in the network. All messages that are sent from one process to another are sent to this network

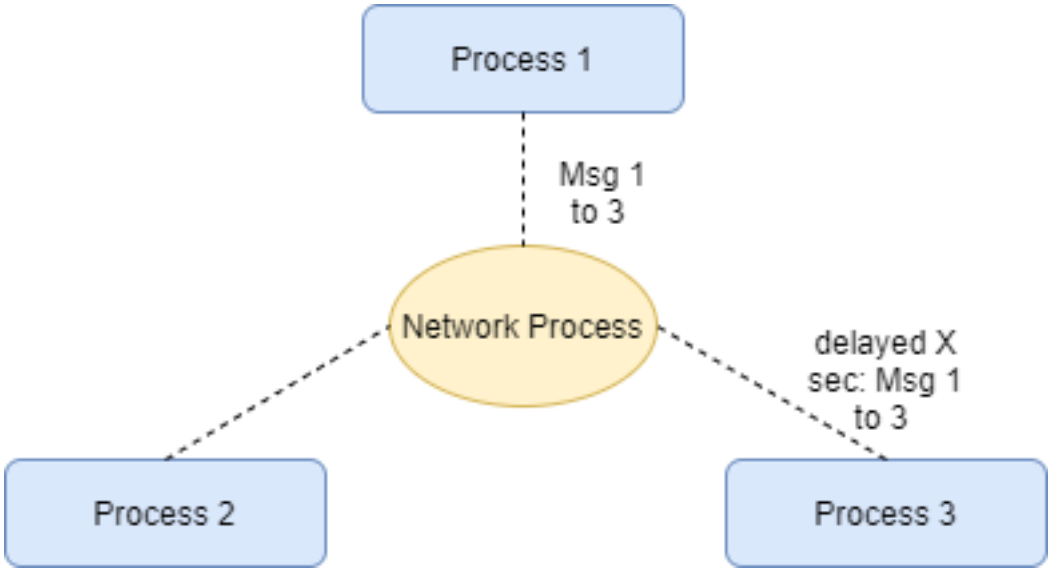


Figure 1: Example

process first. The only functionality of the network process is to **randomly** wait between **1 to 5 seconds** before forwarding the otherwise unaltered message. See Figure 1.

# 3 Implementation details

The events that need to be ordered are life events of different people. Each person is represented by a process and has two types of life events:

- Local events: events such as sleep, eat, shower, wake up, study, play, etc., and
- Communication events: communication between 2 processes by sending a message. The message can be any text (e.g., "hello").

Each process must be uniquely identified using an id, $\boldsymbol{P_1}$,$\boldsymbol{P_2}$, **and** $\boldsymbol{P_3}$. Each process maintains a Lamport logical clock (i.e, a counter) initialized to zero. For each local event, a process increments its local clock and assigns the clock value to this event.

Communication between processes is implemented using TCP or UDP sockets. Communication between two process is represented by a message send event (in the sender process) and a receive event (in the receiver process). A send event must have the unique id of the receiver and a send event piggybacks the local counter value of the sender in the message. A process has to block on receive events until processing the received message. Upon receiving a message, a process has to compare the local counter value to the received value and decides a value assignment to the receive event and update the local counter accordingly.

Each process is implemented using two threads as shown in Figure 2. A communication thread listens on a socket for any messages sent to this process. Upon receiving a message, the communication thread puts the message in a shared producer/consumer queue and blocks

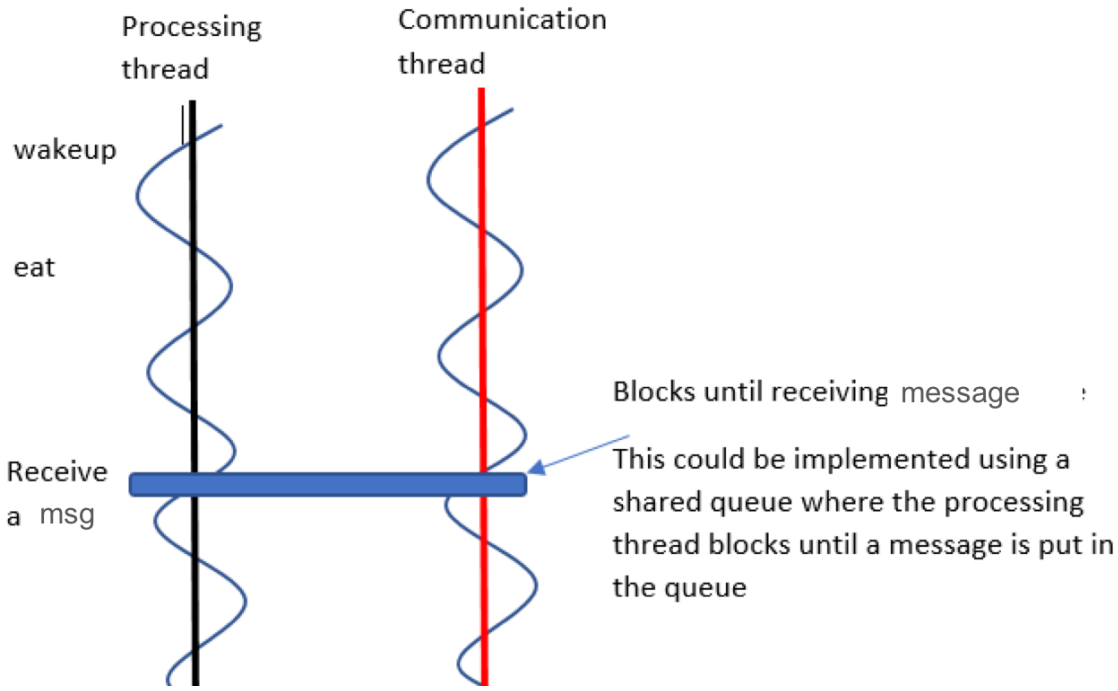


Figure 2: An implementation of a process using 2 threads: a communication thread and a processing thread.

waiting on receiving more messages. A processing thread processes the events one by one until having a receive event. For a receive event, the processing thread pops a message from the producer/consumer queue, if any exists, or block until a message is inserted in the queue. This prevents a potential race condition between the send and receive events.

# 4 System Configuration and User Interaction

NOTE: We do not want any front end UI for this project. All the processes will be run on the terminal and the input/output for these processes will use stdio.

1. Each server has a unique id $\boldsymbol{P_1}$,$\boldsymbol{P_2}$, **and** $\boldsymbol{P_3}$. When starting a server, it should connect to the network processor and the network processor should know the ports on which the 3 processes are listening. This could be done via a configuration file or other methods that are appropriate.

2. Each process must take 3 types of input from the user:

   - *Local event*: If the user chooses this input, the process should prompt the name of the event.
   - *Communication event*: If the user chooses this input, the process should prompt the receiver id; the process then sends a message along with its local clock value.

- *Print clock*: If the user chooses this input, the process should print a space separated history of the clock for that process, where the last value is the current clock value. The print clock input does not change the value of the Lamport clock.

3. You are welcome to log any other relevant messages on the console during the demons, such as when a process receives a message from another process.

4. Use message passing primitives TCP/UDP for inter-process communications. You can decide which alternative and explore the trade-offs. We will be interested in hearing your experience.

5. There is no restriction on the programming language.

# 5 Sample Scenario

Sample inputs for 3 processes and a potential execution are shown below. Ex. 1 is a simple example with only 1 message, and Ex. 2 is more involved.

## 5.1 Ex. 1

$\boldsymbol{P_1}$ local event: "Wakeup"
$\boldsymbol{P_2}$ send event ("Hello"): Receiver ID $\boldsymbol{P_1}$
$\boldsymbol{P_1}$ local event: "Eat"
$\boldsymbol{P_1}$ receive event: receive message ("Hello") from $\boldsymbol{P_2}$
$\boldsymbol{P_1}$ local event: "Sleep"
$\boldsymbol{P_3}$ local event: "Play"

Note, that due to an arbitrary delay in the network, $\boldsymbol{P_1}$ receives the message after the local "Eat" event, but the message could have also arrived before "Eat" or after the "Sleep" event.
$\boldsymbol{P_1}$ Print clock: 1 2 3 4
$\boldsymbol{P_2}$ Print clock: 1
$\boldsymbol{P_3}$ Print clock: 1

## 5.2 Ex. 2

$\boldsymbol{P_1}$ local event: "Wakeup"
$\boldsymbol{P_1}$ local event: "Shower"
$\boldsymbol{P_3}$ local event: "Wakeup"
$\boldsymbol{P_1}$ send event: Receiver ID $\boldsymbol{P_3}$
$\boldsymbol{P_2}$ local event: "Wakeup"
$\boldsymbol{P_3}$ receive event: receive message ("Hello") from $\boldsymbol{P_1}$
$\boldsymbol{P_1}$ local event: "Eat
$\boldsymbol{P_2}$ send event: Receiver ID $\boldsymbol{P_1}$
$\boldsymbol{P_1}$ receive event: receive message ("Hello") from $\boldsymbol{P_2}$

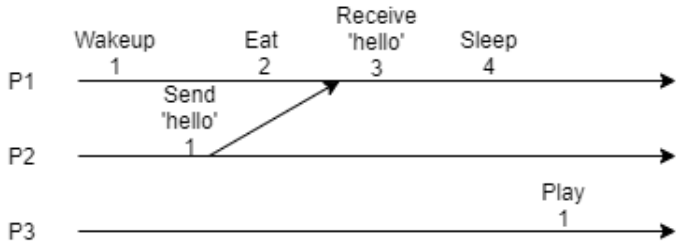


Figure 3: Lamport diagram for Ex.1

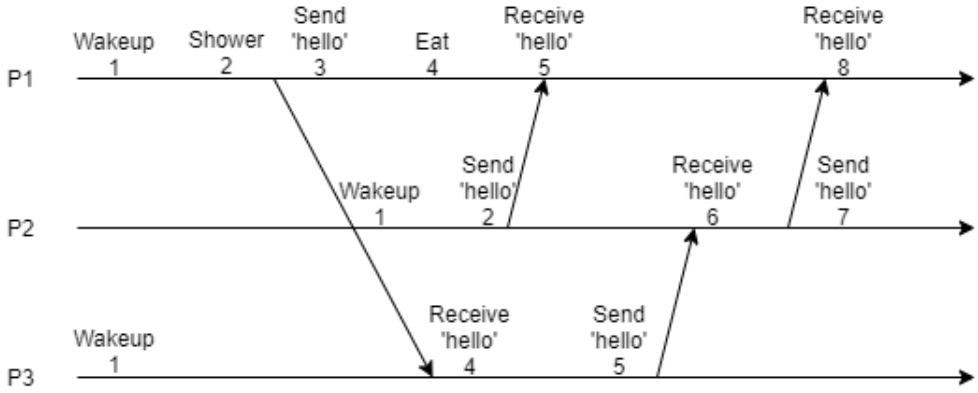


Figure 4: Lamport diagram for Ex.2

$\boldsymbol{P_3}$ send event: Receiver ID $\boldsymbol{P_2}$
$\boldsymbol{P_2}$ receive event: receive message ("Hello") from $\boldsymbol{P_3}$
$\boldsymbol{P_2}$ send event: Receiver ID $\boldsymbol{P_1}$
$\boldsymbol{P_1}$ receive event: receive message ("Hello") from $\boldsymbol{P_2}$

$\boldsymbol{P_1}$ Print clock: 1 2 3 4 5 8
$\boldsymbol{P_2}$ Print clock: 1 2 6 7
$\boldsymbol{P_3}$ Print clock: 1 4 5

# 6 Demo

We will have a short demo for this project. It will be on **April 17th, 2020** via Zoom, and both team members should be present. Time details will be announced later. Please be ready with the working program at the time of your demo.