Global Snapshots in a Distributed Economy

1. Introduction

The Chandy–Lamport algorithm is a snapshot algorithm that is used in distributed systems for recording a consistent global state of an asynchronous system. [1] The algorithm records a global state which includes each process as well as the communication channels.

You have been been hired by the city of Lamportia to help them take snapshots of their economy. Each citizen has a bank account which they can use to send money to other citizens. The following rules have been specified:

1. The government wants to make sure there is no fraud, so every snapshot should add up to the same total sum. IE. if there is 1000 dollars in the system at the start, every global snapshot should add up to 1000 when considering local balances and channels.
2. All communication channels are **FIFO** and **Unidirectional**
3. Any citizen should be able to initiate a snapshot at any time in order to make sure the economy is working fairly
4. A snapshot consists of the local state as well as the state of all incoming channels
5. The snapshot process shouldn't interfere with normal trade and should be consistent

2. Implementation

Each citizen is represented as a process which we will simply refer to as a **site**. They start with a bank account balance of 10. All values will be in integers.

There will be **two input files**: one **setup** file as well as one **command** file.

The setup file for one test scenario will be same for all sites. The first line of the setup file is an integer which represents the total number of sites which we will call **N**.

The next **N** lines will consist of the IP and port of each site separated by a space, sorted by ID.

The rest of the lines in the setup file specify the unidirectional links between sites with two integers separated by spaces. Ie **{site_id_from} {site_id_to}**

Every process should begin by reading the setup file and opening links that correspond to its site as TCP sockets. We only want to open these sockets once at startup, to ensure they are FIFO. At the same time it should accept all incoming connections and keep them open as well.

The **command** file consists of the following commands:

- **send {site_id} {amount}** send money from local balance to site_id
- **snapshot** starts the snapshot algorithm
- **sleep {time}** stops the process for the given number of seconds. Every 200 ms you should wake up from your sleep and check your incoming sockets and receive all messages from them and handle them. A 1 second sleep would consist of waking up for 5 x 200 ms intervals.

NOTE: There is no explicit receive command. Instead you should receive messages and markers from the TCP socket whenever they are available. This means checking all your incoming communication streams before and after processing every command and reading all incoming messages. (if there are 2 messages in stream, read them both, if there is 0, don't block)

The communication channels should remain open until all events are processed. **There is no need to use multi-threading in this assignment.**

The site_id as well as input files of the process should be taken in as command line arguments. Here is the format:

./asg2 {site_id} {setup_file} {command_file}

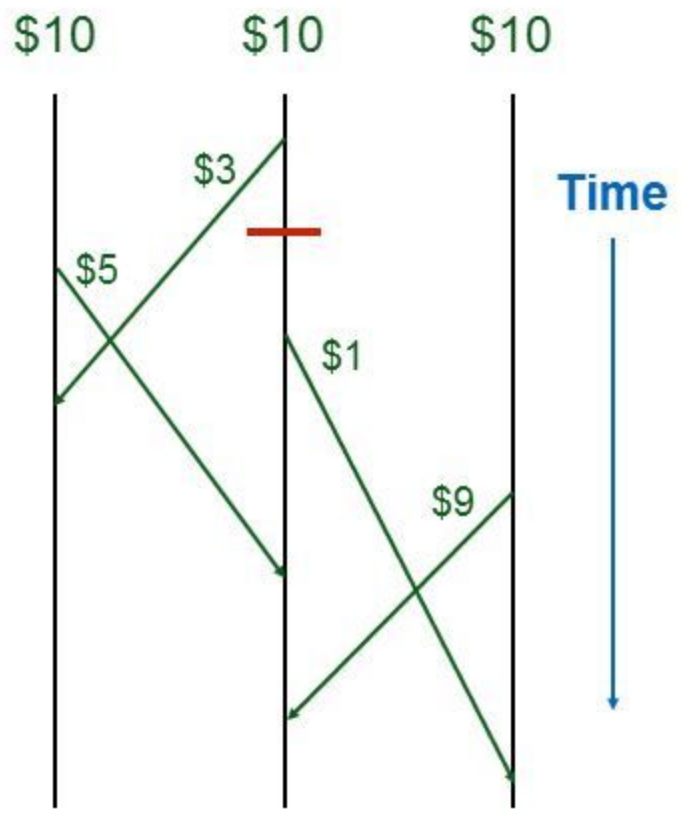
site_id is the id assigned to the site (start with index 1). Each process will be started with the same command varying only in site_id and command file.

Each site should output the result of each snapshot line by line. Each line should consist of one snapshot. A snapshot consists of a snapshot_id and the local state followed by each communication channel's state ordered by site_id. The snapshot_id is obtained from concatenating the site who started the snapshot with the count of the snapshot separated by a (**.**) e.g. snapshot 2.3 is the third snapshot started by site 2.
Example format for snapshot from site1 w/ 3 sites:

{snapshot_id}: {local balance} {site2 channel} {site3 channel}

3. Sample Input/Output

SCENARIO



COMMAND FILES

Site 1 (site1.txt)	**Site 2 (**site2.txt)	**Site 3 (**site3.txt)
send 2 5	send 1 3	send 2 9
	snapshot	sleep 5
	send 3 1	

SETUP FILE (setup.txt)

3
127.0.0.1 5001
127.0.0.1 5002
127.0.0.1 5003
1 2
2 1
2 3
3 2

START COMMANDS

P1	P2	P3
./asg2 1 setup.txt site1.txt	**./asg2 2 setup.txt site2.txt**	**./asg2 3 setup.txt site3.txt**

OUTPUT

P1	P2	P3
2.1: 8 0 0	2.1: 7 5 9	2.1: 1 0 0

Note that the sum of all sites output is 30! This is the same as the start where each site had 10. However, notice that this output is not unique but irrespectively, the sum is always 30.

4. Turnin

You may work in **two person teams** on this assignment. There is no restriction on the programming language but you have to use a platform that can run on any of the Linux CSIL machines. You should submit one tar file on Gaucho space that has a README, the source files, and an executable file that can run directly on a Linux CSIL machine by **May 8th** 11:59pm. Please make sure to name your executable **asg2**. We will run this executable file over some test cases and give you credit based on your program responses. There is no partial credit on submitting non functioning code so it is better to start NOW.

[1] https://en.wikipedia.org/wiki/Chandy-Lamport_algorithm