UNIVERSITY OF TARTU
Institute of Computer Science

Distributed Systems (Spring 2021)

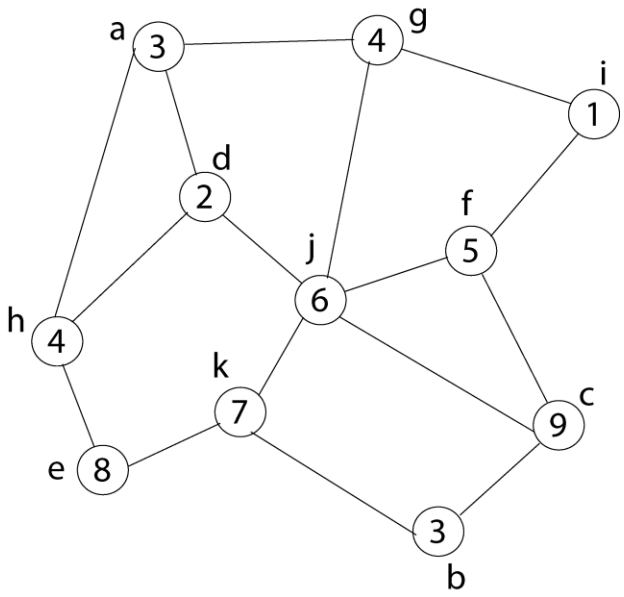
Task 2: clock synchronization and election algorithms

Practical information:
Due date: Monday, April 12, midnight (FIRM DEADLINE)
- This task can be performed in a team of max 3 persons ONLY.
- Total amount of pts that that can be collected = 100 + Bonus pts = 123 pts. **Bonus pts are optional**

Submit your solution to farooq.ayoub.dar@ut.ee, mohan.liyanage@ut.ee, (CC) huber.flores@ut.ee

Instructions: Please respond to the following questions. **Be clear and concise in your answers.** Provide enough explanation to support your arguments. Ambiguous answers to fill up space are considered wrong and no points are granted. For questions that involve implementation, be sure to include everything needed to execute your program (Re-submissions are not allowed).

1. Consider the following wireless network (sensor network). Assuming that source (g) starts a simple election for a leader. What is the actual path used to inform the source about the most suitable leader? Illustrate the election process with figures in each step. (20 pts)



2. By using different amount of processes (1000, 10000, 100000, 1000000), compare the performance of using election in a ring vs election by bully for selecting a coordinator. Model the performance of each algorithm by estimating the number of messages sent in each case. No implementation is needed (20 pts)

3. Write a program implementation that combines the bully algorithm with the Berkeley algorithm for clock synchronization. (60 pts)

Berkeley algorithm uses a central coordinator to synchronize clocks between processes when there is not access to external clocks. Examples of systems using Berkeley algorithm, include, distributed development environments (compiler and editors) and local area network functionalities, e.g., sharing a resource (printer)

Bully algorithm selects a coordinator among a list of processes. The process with the highest id always obtains the coordinator position. Examples of systems using Bully algorithm, include distributed databases (selecting primary database) and distributed computing (selecting master and slave devices)

Bully + Berkeley: We will implement the fundamental logic of the algorithms to coordinate and synchronize activities in a decentralized system of processes. The main idea is that a list of processes synchronizes their clock using a coordinator. If the coordinator fails, then a new coordinator is selected using bully algorithm. Once the new coordinator is selected, all processes synchronized their clock based on the new coordinator time.

Implementation requirements: It is possible to use anything in your implementation, but algorithm logic should be highlighted.

Specifications
Command line interface:

$ clock-sync-by-bully ***processes_file***

Where,

1. *clock-sync-by-bully* is your program
2. ***processes_file*** is the path to the file where processes are defined.

The processes_file has N number of lines, each in the following format separated by commas.
Process_ID, Process_NAME, Process_TIME
1, A_K, 11:00am
3, B_K, 13:33am
4, D_K, 17:30pm
7, E_K, 23:00pm
5, F_K, 3:00am

Processes defined in the processes_file have unique Ids. When running your program for the first time, the processes_file is loaded into the program, and the bully algorithm is applied by default. A coordinator is selected from this first election (the process with the highest ID must be elected as the coordinator always). An automatic notification should announce the process that is the coordinator after program has started.

When an election takes place, each process NAME is updated. This means that the K value should be increased by 1 each time and election takes place. K is equal to zero when loading the processes. So, the actual processes_file input looks like this.

1, A_0, 11:00am
3, B_0, 13:33am

4, D_0, 17:30pm
7, E_0, 23:00pm
5, F_0, 3:00am

The first election that takes place by default do not increase the election counter. This means that after processes are loaded and a coordinator is elected, the K value should not be increased.

After the program is loaded, a terminal should be available to introduce commands to the program. The following commands should be implemented.

$ List (5 pts)
It shows the list of processes running, and indicate the process that is acting as coordinator.

1, A_0
3, B_0
4, D_0
7, E_0 (Coordinator)
5, F_0

$ Clock (5 pts)
It shows the current clock time of each process (Clocks are synchronized based on coordinator clock – this is for simplicity; we won't take the time average of each process as stated by Berkeley)

A_0, 23:00
B_0, 23:00
D_0, 23:00
E_0, 23:00
F_0, 23:00

$ Kill Process_Id (10 pts)
This command allows removing a process from the program (including the coordinator). If the coordinator is removed, clocks are reset to their original time (the one defined in the process_file), and a new election needs to happen automatically. The program should notify the process that is the new coordinator, and which process started the election. Clocks of all the processes are synchronized based on the clock of the new coordinator.

$ Set-time Process_id (10 pts)
This command allows setting a new clock time for a particular process (including the coordinator). If the clock of the coordinator is changed, then all processes need to be synchronized again using that time. If the time of any other process is changed, then that process should be synchronized again based on the time of the coordinator (automatically after some time).

$ Freeze Process_id (5 pts)
This command suspends a particular process. When a process is suspended, clock stops and cannot communication with others. Any process can be suspended.

$ Unfreeze Process_id (10 pts)

This command resumes a process that was previously suspended. Once a process is resumed, its clock is synchronized based on the clock of the coordinator. If the suspended process has the higher id in the group, then the process should start bullying others until becoming the coordinator. Once it becomes the coordinator, it uses its default clock as the new time (the time from the process_file). After this, other processes have to synchronize their clocks with it.

$ Reload (15 pts)
This command reloads the processes defined in the file (processes_file) to the running program. If a process already exists, then the process should not be reloaded. If a process was removed (killed), then the process is reloaded again into the program. Once processes are reloaded, a new election should take place. Notice also that new processes can be defined in the input file. Moreover, notice also that the K value of each process should reflect the number of elections in which a process has participated. For instance, if A_0 and B_0 were loaded initially, and participated in two elections, then their current status is A_2 and B_2, respectively. If the process A is then removed, and then there is a new election, the value of B then becomes B_3. If the process A is reloaded (after reload command is applied), and then there is a new election, A is back to the processes pool, and their current values for processes A and B should be A_0 and B_4, respectively.

Deliverables for your clock-sync-by-bully:

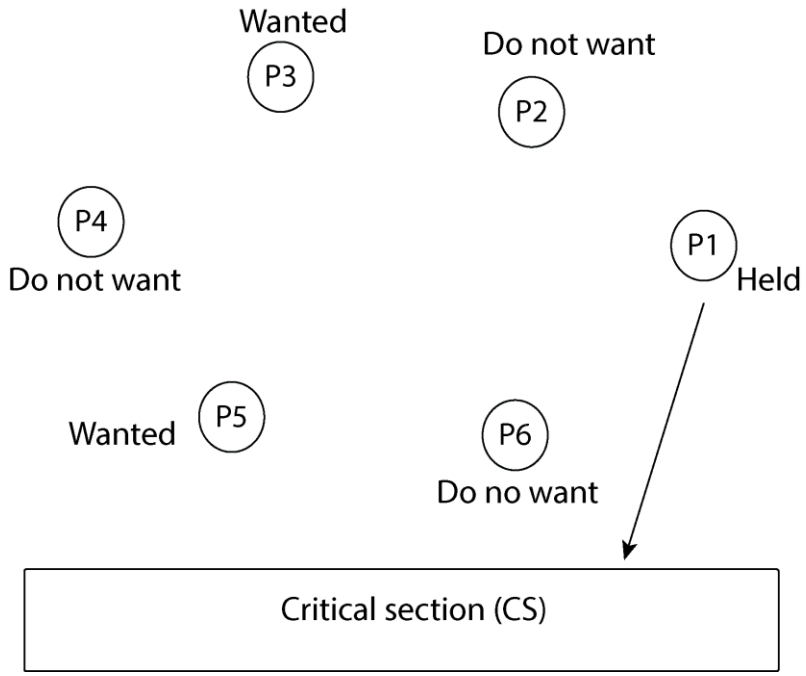
1- A short 1 min video showing your program in execution (showing all the commands)
2- Source code and everything needed for executing your program (binaries and input file). We will evaluate your work with multiple file configurations. Thus, do not hard code your solution just to work with the input file described here.

4. **(5 pts) BONUS POINTS:** In seminar 5, you explored code migration with sockets and java reflection. In that exercise, client and server are communicating in a synchronous manner. Describe the changes that need to be done, such that the client and the server can communicate using asynchronous communications. Extend the exercise (submit your code as part of the deliverables), such that asynchronous communication can be used instead of a synchronous one. Compare both communications and describe their pros and cons. If you don't want to use seminar 5 as code base, you are free to re-implement the whole solution (synchronous and asynchronous) using any language of your choice.

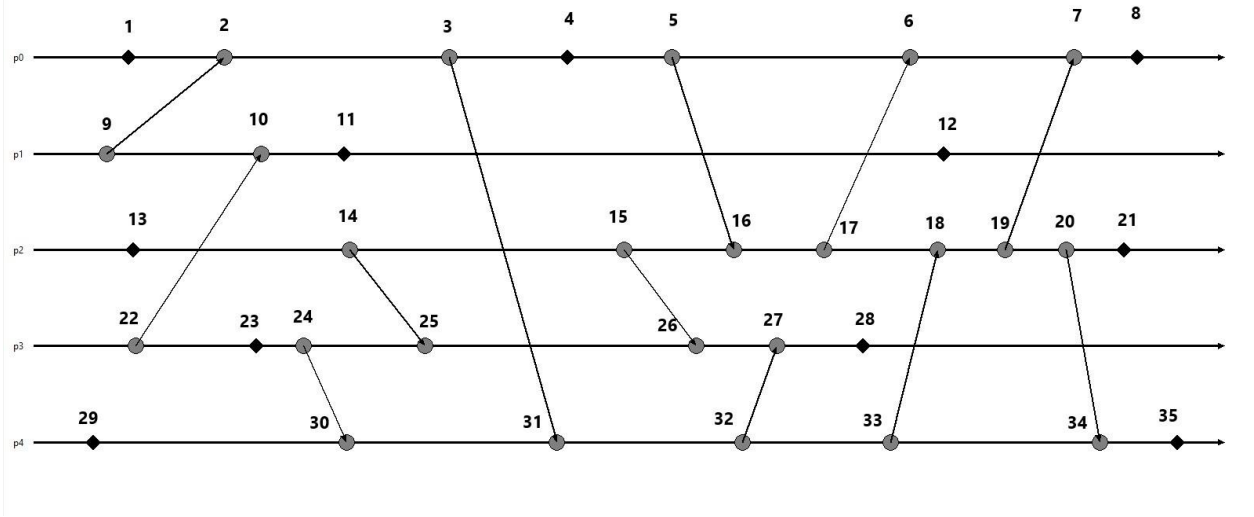
5. **(10 pts) BONUS POINTS:** Nodes (Processes, Resources, Components, etc) of a centralized and decentralized systems rely on mutual exclusion algorithms to access resources in an exclusive manner. Use any language of your choice and implement the Ricart & Agrawala algorithm for mutual exclusion. The implementation of this algorithm will be free, but will be based on the description based on slide 11 of the Lecture 7 – Coordination II. This is to avoid implementing slightly different versions of the same algorithm. The basic idea of the algorithm is as follows, a group of processes sharing a critical session (resource), need to exchange messages with (Lamport) timestamps to coordinate exclusive access to a resource (see Figure below). The one with the lowest timestamp (in case of a process is waiting to enter the CS) is the one that is granted access to the CS. A process can have three different states when coordinating mutual

exclusion, HELD (The process has locked the CS), WANTED (The process wants access to the CS), and DO-NOT-WANT (The process is not interested to enter the CS – but can get want access later on)

a. The input of the program is a list of processes, e.g., [P1, P2, P3, P4, P5]. The program should run continuously and has to provide a way to select the process or processes that want to access the critical section. For simplicity, the time that a critical section is HELD by a process is fixed, e.g., 30 seconds.
b. A short video describing your program in execution (explain it in detail and showcase its implementation by describing your code)
c. Describe whether it is possible that starvation or deadlock situations can arise. (5 pts)



6. **(8 pts) BONUS POINTS:** Consider the *Figure* below that shows five processes (P0, P1, P2, P3, P4) with events 1, 2, 3 ,... etc., and message events communicating between them. Assume that initial logical clock values are all initialized to 0
    - (3 Points) List the Lamport timestamps for each event shown in the Figure below. Assume that each process maintains a logical clock as a single integer value as a Lamport clock. Provide timestamps for each labeled event.
    - (3 Points) List the Vector Clock timestamps for each event shown in the Figure below. Provide timestamps for each labeled event.
    - (2 Points) List the events that are concurrent and explain in detail why? Is there the potential for a causal violation? Reflect on both Lamport and Vector Clock algorithms and explain whether both approaches can be used to accurately identify concurrent events.


p0
p1
p2
p3
p4
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35