CMSC 621: PROJECT 2

CLOCKS, MULTICAST, AND COMMIT

Name: Intekhab Naser **ID: ZC11577**

README:

Instructions on compiling and running the code base:

Assignment - 1:

In this Assignment, there are two programs: a server program which acts as Time Daemon and a client program which consults the Time Daemon to adjust their clocks.

1) **Compiling the program:**

To compile the server and client files, I use Makefile as follows:

```
compile: server.o client.o
server.o: Berk_s1.cpp
    g++ -o server Berk_s1.cpp -lpthread
client.o: Berk_c1.cpp
    g++ -o client Berk_c1.cpp -lpthread
clean:
    rm -rf server client
```

The following command compiles both the files:

⇨ Make compile

The following command deletes the object files:

⇨ Make clean

2) **Running the program:**

It should be taken into consideration that, while running the program, only one server process should be executed, while multiple client processes may be connected to server.

To execute server program, use the following command:

⇨ ./server <port number>
  - After this, Server (Time Daemon) asks:

⇨ "Enter the total number of machines to connect: "
  - We should specify the number of clients we intend to connect

To execute client program, use the following command:

⇨ ./client <port number>

Once, all the clients are connected, the Berkeley's Algorithm starts.

-x-

Assignment - 2:

In this Assignment, there are two versions of the Multicast Program:

(i) One program is implementation of Multicasting messages with Causal Ordering
(ii) Other program is also implementation of Multicast messages but is Non-Causal

In these programs, all the processes may behave as server and client both, as needed.

1) Compiling the program:

To compile both the programs: Causal and Non-Causal, I use Makefile as follows:

```
compile: causal.o non_causal.o
causal.o: Multicast_Causal.cpp
    g++ -o causal Multicast_Causal.cpp -lpthread
non_causal.o: Multicast_NonCausal.cpp
    g++ -o non_causal Multicast_NonCausal.cpp -lpthread
clean:
    rm -rf causal non causal
```

The following command compiles both the files:

⇨ Make compile

The following command deletes the object files:

⇨ Make clean

2) Running the program:

There are some specific considerations which needs to be taken care of while running any of these programs.

There is a file in the respective directory of this assignment called “ProcInfo.txt”. The total number of processes/machines to be executed for processing the program is taken from this file. You may update this file with the desired number before starting to run the program. (eg. 4).

To execute program, use the following command:

⇨ ./causal <port number>

Or, When processing the non-causal program, use this command:

⇨ ./non_causal <port number>
  - Note that, all the processes must use different port numbers from each other

Now, as you run the program. Each process takes some information from the user:

⇨ 1) What is your process ID?
⇨ 2) Do you wish to connect to any machine? (yes = 1 / no = 2)
⇨ 3) Enter the number of machines to connect:
⇨ 4) Enter the port number of Machine to connect:

The process is explained with an example of 4 processes:

Let’s assume that the ProcInfo.txt contains the number ‘4’. So there are total 4 processes in our system.

- ➢ You need to give all the processes, a unique process ID (eg. 1, 2, 3, 4)
- Now for process 1, say yes to connect to any machine. Give the number of machines as (n - 1). i.e 3 in our case.
- For process 2, say yes to connect to any machine. Give the number of machines as (n - 2). i.e 2 in our case.
- For process 3, say yes to connect to any machine. Give the number of machines as (n - 3). i.e 1 in our case.
- For process 4, say no to connect to any machine as the number of machines is (n - 4). i.e 0 in our case
- ➢ Now, follow the same order to connect to all the processes:
- First, all the processes connect to the last i.e. 4th process. Give the port number of process 4 to process 1, then to process 2 and then to process 3. At this point, Process 4 is connected to all other processes.
- Now, all remaining processes connect to the second last i.e. 3rd process. Give the port number of process 3 to process 1 and then to process 2. At this point, Process 4 is also connected to all other processes.
- Finally, the remaining process connects to third last i.e. 2nd process. Give the port number of process 2 to process 1. At this point, all the processes are connected to all other processes.
- ➢ Now all connections are completed. Follow the above procedure for any number of processes.
- ➢ At any process, enter 1 to multicast messages 100 times.

<u>**Bonus Assignment**</u>:

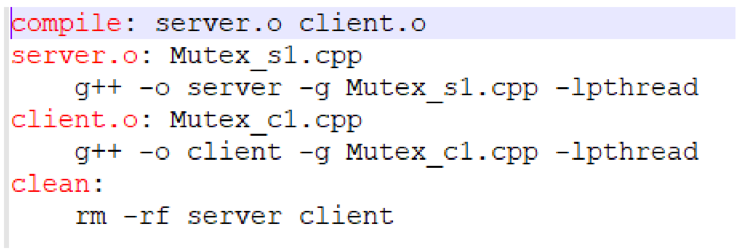
In this Assignment, there are two programs:

(i) A server program which acts as Centralized Coordinator to access shared resource in order to manage distributed locking mechanism for accessing a shared file between clients.

(ii) A client program which consults the Coordinator to read counter from the shared file, update it and write back.

1) Compiling the program:

To compile the server and client files, I use Makefile as follows:



```
compile: server.o client.o
server.o: Mutex_s1.cpp
    g++ -o server -g Mutex_s1.cpp -lpthread
client.o: Mutex_c1.cpp
    g++ -o client -g Mutex_c1.cpp -lpthread
clean:
    rm -rf server client
```

The following command compiles both the files:

⇨ Make compile

The following command deletes the object files:

⇨ Make clean

3) Running the program:

It should be taken into consideration that, while running the program, only one server process should be executed, while multiple client processes may be connected to server.

To execute server program, use the following command:

⇨ ./server <port number>
  - After this, Server (Centralized Coordinator) asks:

⇨ "Enter the total number of machines to connect: "
  - We should specify the number of clients we intend to connect

To execute client program, use the following command:

⇨ ./client <port number>

As soon as any client connect to the server, it can initiate the REQUEST to access the shared file.