Zack Campbell - zcc254
Kazuhiro Aoyama - ka25635

# HW3

1. Some applications require two types of accesses to the critical section – read access and write access. For these applications, it is reasonable for multiple read accesses to happen concurrently. However, a write access cannot happen concurrently with either a read access or a write access. Modify Lamport's mutex algorithm for such applications.

Assume Logic Clock algorithm running

```
Var   Q: queue of (int, pid) initially null;
      numAcks = 0; initially;
      N = number of processor in this application

Request to enter CS for reading:
      Send read request with (timestamp, Pi) to all other processes
      Insert (timestamp, Pi) in q
      numAcks = 0

Request to enter CS for writing:
      Send write request with (timestamp, Pi) to all other processes
      Inser (timestamp, Pi) in q
      numAcks = 0

Upon receiving a request message
      Insert the request (timestamp, Pi) in q
      Send acknowledgement to Pi

Upon receiving acknowledgement:
      numAcks = numAcks + 1

Pi can enter the CS for reading if:
      numAcks = N -1 and the timestamp of Pi's request is smallest or all requests before
      the Pi's request are read requests and they receive acknowledges from other processes,
      then Pi enter critical section to read

Pi can enter the CS for writing if:
      numAcks = N -1 and the timestamp of Pi's request is smallest, then Pi enter critical
      section to write
```

```
  Release:
        delete the request by Pi from q
        Send release message to all processes
```

2. (a) Extend Lamport’s mutex algorithm to solve k-mutual exclusion problem which allows at most k processes to be in the critical section concurrently.

Assuming Lamport’s clock is running and the principle of FIFO and perfect failure detection, Lamport’s mutex algorithm can be extended to allow k processes into the critical section concurrently. The following algorithm implements this extension:

```
Var numAcks: int init 0

To request CS:
        Send a timestamped request to ALL processors

Upon receiving a request message:
        Put this request in the queue q ordered by timestamps
        Send an acknowledgement to that process

Process Pi can enter CS if:
        It’s request is in the top k places in the queue
        It has received acknowledgement from all processes

Upon receiving an acknowledgement:
        Increment a counter: numAcks++

Upon receiving a release message:
        Delete that request from the queue
```

(b) Extend Ricart and Agrawala’s mutex algorithm to solve the k-mutual exclusion problem.

Ricart and Agrawala’s mutex algorithm is very similar to Lamport’s mutex algorithm but optimizes it by combining the acknowledgement and release messages into a single “okay” message. An algorithm expanding on R&A’s that solves the k-mutual exclusion problem is as follows:

```
Var     numInCS: int init 0
        numOkay: int init 0

To request CS:
        Send a timestamped request to ALL processors
```

Upon receiving a request message:

Send an “okay” message if the current process is not interested in the CS or if it has a higher timestamped value and numInCS >= k, otherwise append the process that requested to the pending queue

To enter the CS:

It has requested access to the CS and received “okay” messages from ALL other processors

Increment counter upon entering: numInCS++

Upon releasing the CS:

Send an “okay” message to all processes in the queue

Decrement counter: numInCS--

Upon receiving an “okay” message:

Increment counter: numOkay++