Pooja Baba
#002677117
10.31.2022

CSC 8980
Distributed Systems Fall 2022
Homework #3

Question 1 – Mutual exclusion algorithm for logical clocks and centralized resource controller.

1. P_i process will send a *request(t)* (*t* is the logical clock timestamp for the process) message to P_0 centralized controller to access the critical section
2. P_0 controller will then add the request by P_i to the ProcessWaitQueue along with its timestamp
3. P_0 then uses *inform()* message to inform the P_i process about the status of the critical section i.e, whether the critical section is available or occupied and the P_i position in the ProcessWaitQueue.
4. If the P_j process requests the CS by sending a *request(t)* message to the P_0 controller

   The controller will add the process in the ProcessWaitQueue along with its timestamp *t*
5. If the CS is available to be acquired
   a. Controller P_0 performs a search operation on ProcessWaitQueue to find the process with a minimum timestamp
   b. Controller P_0 pops the process and its details from the ProcessWaitQueue
   c. It sends an *inform()* message to the process just popped
6. The process P_i to which the *inform()* message is sent, receives the message sends an *ack()* message to assure the controller of its availability
7. The controller waits for *x* units of time for the process P_i to send an *acknowledgment*.
8. If the controller does not receive an *acknowledgment* from the process in the *x units* of time,

   It places the process P_i at the end of the ProcessWaitQueue with updated timestamp t
9. Controller then goes to *Step 5.a* performs the activity again
10. If the controller receives an *acknowledgment*, the controller sends a *grant()* message to the process P_i
11. Once the *grant()* message is received, process P_i takes control of the CS
12. After the process has completed performing its task in the CS, it sends a *release()* message to controller P_0.
13. Controller repeats steps from 5 iff -
    a. Critical Section to be acquired is available
    b. ProcessWaitQueue is not empty

Pooja Baba
#002677117
10.31.2022

Question 2 – Formal derivation for the inequality := $\varepsilon / (1 - \kappa) \leq \mu$

ANS:

C_i (t) = reading of clock C_i at physical time *t*

we assume that $C_i(t)$ is a continuous, differentiable function of t except for isolated jump discontinuities where the clock is reset. Then $dC_i(t)/dt$ represents the rate at which the clock is running at time t. We assume the following condition is satisfied –

PC1. There exists a constant $k << 1$ such that for all i: $|dC_i(t)/dt - 1| < k$

(For typical crystal-controlled clocks, $k <= 10^{-6}$*)*

It is not enough for the clocks individually to run at the correct rate. They must be synchronized so that $C_i(t)$ is approximately $C_j(t)$ for all *i*,*j*, and *t*. More precisely, there must be a sufficiently small constant ε so that the following condition holds:

PC2. For all *i, j*: $|C_i(t) - C_j(t)| < \varepsilon$

Let μ be a number such that if event *a* occurs at physical time *t* and event *b* in another process satisfies *a -> b*, then *b* occurs later than physical time $t + \mu$. In other words, μ is less than the shortest transmission time for interprocess messages. We can always choose μ equal to the shortest distance between processes divided by the speed of light.

To avoid anomalous behavior, we must make sure that for any *i, j*, and *t*: $C_i(t + \mu) - C_j(t) > 0$. Combining this with PC1 and PC2 allows us to relate the required smallness of *k* and ε to the value of μ as follows. We assume that when a clock is reset, it is always set forward and never back. (Setting it back could cause C I to be violated.) PC1 then implies that $C_i(t + \mu) - C_j(t) > (1 - k)\mu$. Using PC2, it is then easy to deduce that $C_i(t + \mu) - C_j(t) > 0$ if the following inequality holds:

$$\varepsilon/(1 - k) <= \mu$$

Pooja Baba
#002677117
10.31.2022

Question 3 - solution to the Readers-Writers Problem with writer preference

Algorithm –

(Semaphore: mutex 1, mutex 2 mutex 3, w, r)

READERs:

```
P(mutex 3)
      P(r)
            P(mutex 1)
                  Requesting_Critical_Section := TRUE;
                  readcount_SEQ_NUM = readcount_SEQ_NUM + 1
                  if readcount_SEQ_NUM == 1 then P(w);
                  Outstanding_Reply_Count := N - l;
                  FOR j := 1 STEP l UNTIL N DO IF j != me THEN
                        Send_Message(REQUEST(Our_Sequence_Number, me),j);
                  // sent a REQUEST message containing our sequence number and
our node number to all other nodes;
                  // Now wait for a REPLY from each of the other nodes;
                  WAITFOR (Outstanding_Reply_Count = 0);
            V(mutex 1)
      V(r)
V(mutex 3)
// Critical Section Processing can be performed at this point;
      ...
reading is done
      ...
// Release the critical section
P(mutex 1)
      Requesting_Critical_Section = FALSE
      readcount_SEQ_NUM = readcount_SEQ_NUM - 1
      FOR j := l STEP 1 UNTIL N DO
            Send_Message (REPLY, j);
            // send a REPLY to node j;
      if readcount_SEQ_NUM = 0 then V(w)
V(mutex 1);
```

Pooja Baba
#002677117
10.31.2022

<u>WRITERs</u>:

```
// Request Entry to our Critical Section;
P(mutex 2)
      // Choose a sequence number;
      Requesting_Critical_Section = TRUE
      writecount_SEQ_NUM = writecount_SEQ_NUM + 1
      if writecount_SEQ_NUM = 1 then P(r)
      Outstanding_Reply_Count := N - l;
      FOR j := 1 STEP l UNTIL N DO IF j != me THEN
            Send_Message(REQUEST(Our_Sequence_Number, me),j);
      // sent a REQUEST message containing our sequence number and our node
number to all other nodes;
      // Now wait for a REPLY from each of the other nodes;
      WAITFOR (Outstanding_Reply_Count = 0);
V(mutex 2)
P(w)
// Critical Section Processing can be performed at this point;
      ...
writing is performed
      ...
V(w)
// Release the critical section
P(mutex 2)
      Requesting_Critical_Section = FALSE
      writecount_SEQ_NUM = writecount_SEQ_NUM - 1
      if writecount_SEQ_NUM = 0 then V(r)
      FOR j := l STEP 1 UNTIL N DO
      IF Reply_Deferred[j] THEN
            BEGIN
                  Reply_Deferred[j] := FALSE;
                  Send_Message (REPLY, j);
                  // send a REPLY to node j;
V(mutex 2)
```

Changes Proposed - "readers" never defer a REQUEST for another "reader"; instead they always REPLY immediately. "Writers" follow the original algorithm. This is for the readers writers problem with writer’s preference.

Pooja Baba
#002677117
10.31.2022

Following are the changes proposed to achieve weak/strong reader priority by retaining the Ricart and Agarwal algorithm -

1. When a writer arrives, first check for any readers that are currently holding any lock. If so, the writer shall wait until all readers have released the lock.
2. When a reader arrives, check if there are any writers that are holding any lock. If so, the reader should wait until the writer releases the lock.
3. If a reader arrives while there are other readers waiting for the lock to be acquired, the reader should be given priority over the writer.
4. If a writer arrives while there are other writers waiting for the lock, the writer should be given priority over the readers.
5. Once all the readers/writers have released the lock, the next reader/writer in the line should be given the lock.
6. Once a reader/writer has been given a lock, they should hold the lock for a short period of time so that there is no starvation amongst the other reader/writers
7. Once the reader/writer is done consuming the resource, the lock should be released so that other readers/writers can access the resource.
8. If there are no readers/writers *waiting for the lock* and any reader/writer arrives, the lock to the resource should be granted as soon as the resource becomes available.
9. If there are no readers/writers *currently holding the lock* and any reader/writer arrives, the lock to the resource should be granted immediately.
10. When there are multiple readers and writers waiting for the lock, the readers should be given the priority over the writers.