Lamport's Logical Clocks

Project 1

CPSC 474

19 September 2020

Dr. Doina Bein

David Nguyen 891362089

Summary

For both algorithms calculate and verify, I used threads, condition variables, and priority queue. The calculation algorithm is very simple. Let all threads compute the LC value for all send and terminal events until it reaches a receive event that will need the corresponding send event from other processes. The send event will be stored in a priority queue so that the front of the queue will match any process that has a matching receive event. If all threads have halted their operations or put to sleep, then another sibling thread will tell them to wake up only if there is one process with a receive event that has the earliest encountered search event. Once the process has receive the sent event then it will resume computing the LC value for all send and terminal event until it reaches till a receive event, an empty event, or until there are no more events left to process. Note: that the previous stopping conditions can happen at any time.

The verify algorithm is not so simple. Reverse-engineering the computed values and outputting the correct and incorrect sequences required backward thinking. Determining the terminal and the receive events was pretty simple. However, trying to find the corresponding send event with receive event required a search algorithm. When encountering the receive event, the difference between the preceding event and the encountered received events is greater than 1. Also, a process cannot receive a send event from itself. After tracing through the professor's example in class and in the given guidelines, I noticed that taking the difference between the receiving and sending event would equal to one. Once again, when all the processes has halted by reaching a stopping condition, then a search algorithm will only go search other process that does not belong to the receive event with the lowest LC value and find any search event. Once the search event has been founded it will unlock the process and reconduct the same procedure as described from the beginning until there is a receive event, empty event, or no more events left. Lastly, to determine if the sequence in the process is incorrect will happen if the search algorithm fails to find the corresponding send event.

Pseudocode

Calculate

-Conditions

[x] 1. If a is the first event and is an internal or send event, then LC(a) = 1.

[x] 2. If a is the first event and is a receive event, then LC(a) = k + 1 where k is the LC-value of the send event corresponding to a (that has occurred at a process other than P).
[x] 3. If a is not the first event and is an internal or send event, then LC(a) = k + 1 where k is the LC-value of the event just before a at process P.
[x] 4. If a is not the first event and is a receive event, let b be the send event corresponding to a (that has occurred at a process other than P) and k be the clock value of the event just before a at process P. Then LC(a) = max{ k, LC(b) } + 1

// let a = events
// T = terminal
// S = send
// R = receive

mutex mtx;
condition_variable cv;

vector<pair<string,int>> send;
int max_sleeper = # of n;
int sleeper = max_sleeper;

**Notes:
-will account for the number of processors that needs to finish executing before joining all the processors together.
-if max_sleeper = 0 then all the threads can join and output the result.
-Find the max of both the corresponding LC(send) and the LC value before the receive event then add 1 and return it.

// helper function for condition 4
compareMax(sent, beforeRcv){
if preceding clock value > corresponding send clock value
return preceding clock value + 1

```
  else
   return the send LC value + 1
}

//deadend == a thread with a receive event
calcUnlock(deadend){
  if earliest encountered send event == deadend
          decrement sleeper
          wake up the thread.
  else
          let the thread sleep.
}

//a different sibling thread will supervise the other "processes"
CalcCheckpoint(){
  while there are threads still running
    if all threads are sleeping
      then wake all the threads up.
}

//input is the configuration which contains a list of events
//result contains the computed value will be stored here.
//i is the process id

calcLC(input, result, i){
   k = current clock value.
   loop through the input{
    if(receive event){
      sleeper++;
      put the thread to sleep.
          only wake up when sibling thread says so.
```

```
    k = compareMax(send clock value,  preceding clock value);
    pop the top of the send queue
    fill in the result with the receive clocks value
   }
   else if(send event){
    fill in the result with the send clocks value.
    add the send event to the queue.
   }
   else if(terminal){
    fill in the result with the terminal clocks value.
   }
   else if(empty event){
    fill in the result with 0.
    stop the loop.
   }
  }
  //process is finish.
  max_sleeper--;
}
```

Verify

```
bool found = false
bool isCorrect = false
int max_sleeper = 0;
int sleeper = 0;
int eventCount = 1;
bool findRelatedEvents(){
  loop through the result array
  but do not loop over the current receive process
  find the corresponding send event by taking the difference
  between the current receive and send event.
```

```
}

VerCheckpoint(){
  while there is at least more than 1 process awake.
  if all processes are asleep
    isCorrect = findRelatedEvents();
    if(!isCorrect){
      print “Incorrect”
      exit the threads
    }
    else{
      wake up all the threads
   }
}
VerLC(input, i){
  loop through the input.
  if(receive event) {
    enqueue the receive event onto the front of the queue.
    put the threads to sleep
      only wake up if the corresponding send event has been found
    insert the receive event into the result array
  }
  else if(terminal){
    insert the terminal event into the result
  }
  else if(input == 0){
    insert the NULL into the result.
  }
  --max_sleeper
}
```

Screenshots

README.md - Notepad

File Edit Format View Help

```
# 474-Project-1
Project 1: Lamport's Logical Clocks

Group members:

David Nguyen vietcloudx@csu.fullerton.edu 891362089

To run my file:

On a CLI just type: "sh run.sh" and you are good to go.

If not just click on the main.exe.
```

Type here to search 3:57 AM 10/9/2020

Calculate

```
C:\cygwin64\home\Diana\project-1-lamport-david>sh run.sh
lamport clock is running.

Calculation
---------------
Test Case 01:
1 2 8 9
1 6 7 0
3 4 5 6
---------------
Test Case 02:
1 2 8 9
1 6 7 0
2 3 4 5
---------------
```

Verify

```
---------------
Verification
---------------
Test Case 01:
s1 a r3 b
c r2 s3
r1 d e s2
---------------
Test Case 02:
---------------
INCORRECT
lamport clock has finished running.
```