Carlos Meza

CPSC 474

Professor: D. Bein

CWID: 891577306

Lamport's Logical Clocks

How to use the program:

Note: This was developed with Visual Studio. Windows is the preferred system but still works in a Linux environment.

1. Run program.
2. You will be asked to type in a txt file for input. (Choose from testC.txt or testV.txt)
3. Program will then ask you if you would like to Calculate(C) or Verify(V) the Lamport's Logical Clocks you inputted from the txt file. Make sure input corresponds to the operation you would like to perform.
4. Based on the operation chosen it will output the correct Lamport's Logical Clock values or it will notify you that the input provided was incorrect. It will also output solution to output.txt file.

Shortcomings:

- The program assumes that the receive/send events begin incrementing from 1, however if you have less than 10 sends/receives, then starting at 0 should not cause trouble. In other words, 10 is the maximum number of receives/ sends allowed for the program.
- It is unable to handle sends without receives at this moment in time.
- Also note that internal events should not begin with r or s.

Data Structures

```
Struct Index
{
        String event_type;

        Int LC; // Logical Clock value

        Int xpos;   // row position in matrix

        Int ypos;  // column position in matrix

}
```

Vector<Index> **receives.**

Vector<Index> **sends.**

Vector<Index> **events. //** handles all internal events and NULLs

Functions used:

isNotFull(NxM matrix)

- This checks to see if matrix is completely populated.

printMatrix(NxM matrix)

- This prints a fully populated matrix

sendRecieveMatch(vector<Index> sends Index receive)

- This returns a send Index that corresponds with the receive inputted else it returns an empty Index.

Calculate Algorithm

Convert input text into a string NxM matrix and call it **matrix**.

Make an integer copy of the same size of **matrix** but with empty values and call it **LCMatrix**. (This is the matrix the user will see as output)

From **matrix** we need to process the various events (receive, send, internal/event), and place them in their corresponding vectors.

```
Def calculate():
    While(isNotFull(LCMatrix)
    {
        If(events.size() != 0)
        {
            For tmp in events
            {
                If(tmp.ypos == 0)
                {
                    LCMatrix[tmp.xpos][tmp.ypos] = 1;
                    Events.erase(i);
                }
                Else if(LCMatrix[tmp.xpos][tmp.ypos-1] != EMPTY)
                {
                    tmp.LC = LCMatrix[xpos][ypos-1] + 1
                    LCMatrix[tmp.xpos][tmp.ypos] = tmp.LC +1;
                    Events.erase(i);
                }
                Else if(tmp.event_type == "NULL"}
                    LCMatrix[tmp.xpos][tmp.ypos] = 0;

        }
```

We then process the **sends** in a similar fashion however we do not erase the values just incase there are multiple receives for one send. So, there is no need for the first if statement before we loop through **events**. It also sets the logical clock values of all sends.

```
    If(receives.size() == 1)
    {
        This does the same processing of receives as below, however if it can't find a
        corresponding send it returns INVALID INPUT.
    }
    Else
    {
        For tmp in receives
        If(tmp.ypos == 0)
        {
            Index send = sendRecieveMatch(sends, tmp.event_type);
            If(send.LC != EMPTY)
                Continue;
            Else
            LCMatrix[tmp.xpos][tmp.ypos] = send.LC + 1;
            Receives.erase(tmp);
        }
        Else if(LCMatrix[tmp.xpos][tmp.ypos-1] != EMPTY)
            Int k = LCMatrix[tmp.xpos][tmp.ypos-1]
            Index send = sendRecieveMatch(sends, tmp.event_type)
            If(send.LC != EMPTY)
                Continue;
            Else
            LCMatrix[tmp.xpos][tmp.ypos] = max(k, send.LC) + 1;
            Receives.erase(tmp);
    }
    Output(LCMatrixB)
}
```

Drawbacks

The main draw back of this method is that we have to check if **LCMatrix** is full after every iteration this at worst case will have to O(NxM) * number of times we iterated the while loop.

One way to improve this algorithm is to perhaps find way to locate all the empty positions in **LCMatrix**, store them and loop through them until **LCMatrix** is filled completely or run into an error.

Verify Algorithm:

Convert input text into an integer NxM matrix and call it **LCMatrix**.

Make an string copy of the same size of **LCMatrix** but with empty values and call it **matrix**. (This is the matrix the user will see as output)

From **LCMatrix** we need to process the various events (receive, send, internal/event), and place them in their corresponding vectors.

Events are placed in **receives** if the first element in a processor is not equal to 1. In addition, if there is a gap between the current event and the next one. (Ex. 5   9). The current event will be placed in **sends**, and the next event will be placed in **receives.** All other events will be treated as internal events and placed in **events**. Note, when any event is placed in its corresponding vector, we give it the appropriate LC, xpos, and ypos values.

We then order all vectors by their LC values, from here we are ready to verify if the input can be mapped back to its string equivalent.

```
Def verify()
{
    Bool = found            //used for error checking if there is a receive without a send
    Int msgNum = 1;         //handles the msg counter for send and recv events
    While(receives.size() != 0 && found == true)

        Event recv = receives[0]

        If(recv.LC == 0)

            Matrix[recv.xpos][recv.ypos] = “NULL”
            receives.erase(recv)

        Else

            For sent in sends

                If(recv.LC -1 == sent.LC)
```

```
                    matrix[recv.xpos][recv.ypos] = “r” + to_string(msgNum)
                    matrix[sent.xpos][sent.ypos] = “s” + to_string(msgNum)
                    msgNum++
                    found =
                    receives.erase(recv)
                    sends.erase(send)
                    break
                else
                    found = false

        if(!found)
            for event in events
            If(recv.LC -1 == event.LC)
                    matrix[recv.xpos][recv.ypos] = “r” + to_string(msgNum)
                    matrix[sent.xpos][sent.ypos] = “s” + to_string(msgNum)
                    msgNum++
                    found = true
                    receives.erase(recv)
                    events.erase(event)
                    break
                else
                    found = false
        if(!found)
            Error Message incorrect input
        Else
            Empty sends into events
            Order events based on xpos to get correct order
            Place all internal events into matrix
            Output(matrix)
}
```