COSC 40203 - Operating Systems

Project 5: Lamport and Vector Logical Clocks

Due: 23:59:59, April 29th, 2016 (no extension)

1 Implementing Logical Clocks (70 points)

For this part of the assignment you are to implement Lamport's logical clocks using MPI. You will implement a set of primitives to maintain the state of logical clocks. To show that your logical clocks work correctly, your simulation will read in a set events and then print the logical clocks associated with each event. It is recommended that a manager process read the events and manage the simulation and the remaining processes will update events, exchange messages, and implement the logical clocks.

1.1 Input

The input will be in the following format:

```
<number of processes N>
<event type> <pid1>
<event type> <pid1> <pid2> <msg>
...
end
```

Process IDs in the input file will range from 1 to N. There will be exactly one input event per line. There are two event types that may appear in the input: exec and send. An exec event has no second argument, and indicates that an execution event has taken place in process pid1. A send event denotes the sending of a message from pid1 to pid2. The message will be a string of printable ASCII characters delimited by quotes. The quotes should not be stored as part of the string. Example:

```
3
exec 1
send 1 2 "silly message"
end
```

1.2 Output

At the beginning of the simulation, the manager process will output the number of processes with the following line:

```
printf ("[0]: There are %d processes in the system\n", size);
```

For each event, output the MPI process number, a message describing the event, and the logical clock value.

```
printf("\t[%d]: Execution Event: Logical Clock = %d\n", myrank, logicalclock );
printf("\t[%d]: Message Sent to %d: Message >%s<: Logical Clock = %d\n", myrank, receiverrank,
printf("\t[%d]: Message Received from %d: Message>%s<: Logical Clock = %d\n", myrank, senderran
```

When the manager process reads the "end" from standard input, the manager will instruct each worker process to print it's final logical clock value and then return back to the main program.

```
printf("\t[%d]: Logical Clock = %d\n", myrank, logicalclock);
```

1.3 Example

```
linux> mpirun -np 3 lamport < test1.txt
[0]: There are 2 processes in the system
    [1]: Execution Event: Logical Clock = 1
    [1]: Message Sent to 2: Message >silly message<: Logical Clock = 2
    [2]: Message Received from 1: Message >silly message<: Logical Clock = 1
[0]: Simulation ending
    [1]: Logical Clock = 2
    [2]: Logical Clock = 1
linux>
```

2 Implmenting Vector Clocks (30 points)

For this part of the assignment you are to implement vector clocks using MPI. You will implement a set of primitives to maintain the state of the vector clocks. To show that your vector clocks work correctly, your simulation will read in a set events and then print the vector clocks associated with each event. It is recommended that a manager process read the events and manage the simulation and the remaining processes will update events, exchange messages, and implement the vector clocks.

2.1 Input

The input format will be identical to that in Part 1.

2.2 Output

For each event, you should output the following information. For execution events, print that the event has occurred and the vector timestamp at the process. For a send event, print the relevant information at both the sender and receiver. After each event, you should output any vector clocks updated by the event, and only those. At the end of the run, print the vector clocks of all processes in the system.

2.3 Example

```
linux> mpirun -np 3 vector < test1.txt
[0]: There are 2 processes in the system
    [1]: Execution Event: Logical Clock = [1, 0]
    [1]: Message Sent to 2: Message >silly message<: Logical Clock = [2,0]
    [2]: Message Received from 1: Message >silly message<: Logical Clock = [2, 1]
[0]: Simulation ending
    [1]: Logical Clock = [2, 0]
    [2]: Logical Clock = [2, 1]
linux>
```

3 Miscellaneous

- Start all clocks at time at 0 (Lamport) or [0, 0, . . . , 0] (vector).
- The clock δ, should be the 1 for all MPI processes.
- You MPI program will read from standard input and write to standard output.
- The only MPI functions permitted are `MPI_Init()`, `MPI_Finalize()`, `MPI_Comm_rank()`, `MPI_Comm_size()`, `MPI_Send()`, `and MPI_Recv()`, `MPI_Barrier()`.
- You may use an "acknowledgement" message to ensure that the current action (e.g. message being sent to another process) has concluded before the process controlling the simulation reads the next action.
- The name of your program must be `lamport.c` or `lamport.cpp` and `vector.c` or `vector.cpp` respectively. The name of your executable must be `lamport` and `vector`.

4 Assessment and Grading

This assignment may be completed by teams of two. Your program must be written using C or C++ and use MPI. Comment and document all code submitted. Your program will be tested on `thompson.cs.tcu.edu`. You may do development work on your personal machine but final submissions must compile without errors or warnings and execute without core dumping. Use good programming practices by implementing procedures and functions where necessary. You may use the STL in your solution. This project is worth 100 points.

5 Project Deliverables

1. Follow the project submission guidelines.
2. Follow the project documentation standards.
3. Your project must have a Makefile and a README file.
4. Submit your .zip file to the dropbox on Pearson Learning Studio.