Advanced Operating System Project I Report

Name: Meng-Ju Lu
ID: 2021323210

1. Introduction

In this project, we implement the Ricart-Agrawala algorithm for distributed mutual exclusion, with the optimization proposed by Roucairol and Carvalho.

2. System Structure

- The system consists of ten nodes numbered from zero to nine, and each node connects to all other nodes through the Internet (reliable socket connect, TCP).
- Each node will request to enter critical section (CS) for 40 times.

3. Data Structure

In each node, it maintains the following global variables:

About this node:
- **myid**: to identify each node
- **server_sock**: the socket bind to this node
- **seq_no**: to implement Lamppost's clock
- **highest_seq_num**: record the highest sequence number known by this node
- **num_message_send**: record how many messages sent by this node
- **num_message_recv**: record how many messages received by this node
- **request_count**: record how many REQUESTs made by this node per critical section
- **reply_count**: record how many REPLYs made by this node per critical section
- **using_CS**: indicate whether this node is using critical section
- **waiting_CS**: indicate whether this node is waiting for critical section
- **all_nodes_connected**: indicate whether every node is connected with each other
- **received_all_reply**: indicate whether this node has received all REPLYs to enter critical section
- **exit_session**: indicate whether the node has completed its entrances to critical section

About other node:
- **active_connection[i]**: indicate whether node i connects to this node successfully
- **serv_addr[i]**: the addresses of node i
- **reply_from_node[i]**: indicate whether this node has received REPLY from node i
- **defer_node[i]**: indicate whether this node defers the REPLY for node i
- **complete_node[i]**: indicate whether node i has completed its session
- **portno[i]**: the poor number of node i
- **sockfd[i]**: the socket used to communicate with node i

Since a node communicates with other node by message, we have to define the data structure of message should include the following three data:

- **type**: the type of message (REQUEST, REPLY, COMPLETE)
- **my_id**: ID of the sending node
- **seq_no**: sequence number of the message

4. Library

- POSIX Thread, C++ STL, C Socket

5. Implementation

In each node, it has to do two tasks: receiving REQUEST, REPLY and COMPLETE message and sending REQUEST message. We can separate these tasks to different threads:

- A thread is used to send REQUEST to other nodes for entering critical section, and COMPLETE message to exit the session.
- Other threads are responsible to handle the connections with other nodes (one thread, one node).

For sending and receiving message,

- When sending message, we will serialise the message structure into string to insure it will be sent correctly by **messageSerialization**.
- When receiving message, we deserialise the message into message structure by **messageDeserialization.**

In terms of socket,

- one socket serves as server
- nine sockets serve as clients

6. Result

Node#	Total Sending Message	Total Receiving Message
0	569	569
1	567	567
2	572	571
3	575	574
4	572	571
5	577	576
6	572	571
7	575	574
8	573	572
9	578	577

7. Note

- Some nodes' message exchanges and elapsed time decrease in the end of the session because some node will end its session first, for this early ended node, we don't have to send REQUESTs to these nodes.