Aayush Kumar
CO21BTECH11002
Assignment 2

Roucairol-Carvalho algorithm:

Network description:
- Each process has a socket to receive messages from other processes, called receiver
- Each process has a list of sockets to send messages to other processes, called senders
- Each process has a base endpoint, succeeded by the process id, to create a unique endpoint for each process

procVars class:
- willReceive: list of processes which will receive the request
- deferred: list of processes whose request is deferred
- wantsCS: status of process - whether it wants to enter CS
- inCS: status of process - whether it is in CS
- numRepliesExpected: number of replies expected to enter CS
- numRepliesReceived: number of replies received
- lamportClock: current lamport clock
- csEntryClock: clock at which process requested to enter CS
- eventMtx: semaphore to protect the process variables

reqCS(): request to enter critical section. Sets wantsCS to true, increment lamport clock, sets csEntryClock to current lamport clock

canEnterCS(): check if the process can enter the critical section. Possible only if number of replies received is equal to number of replies expected

enterCS(): enter critical section - set wantsCS to false, set inCS to true

exitCS(): exit critical section - set inCS to false

receiveReply(): update number of replies received

init():
Initializes the process variables and sockets. For the first request, all processes which have id less than the current process will receive the request.
logEvent(string message):
Logs the event or sends the message to the parent process.

sendReq(string message, int receiver):
Sends messages to other processes. If receiver = -1, send a request to all other processes that will receive the request. Otherwise send a request to a specific process specified by the receiver.

sendReply(string message, string receiver, bool deff):
Send a reply to other processes. If deff is true, send a reply to all deferred processes. Otherwise send a reply to a specific process specified by the receiver. Once a reply is sent to a process, say p_j, p_j is involved in critical section executions and hence will receive the request later from the current process. So, the willReceive list is updated.

doWork():
Performs the local computation, requests to enter critical section, waits for replies, enters critical section, does some work in critical section, exits critical section, sends reply to all deferred processes. The amount of time spent in the critical section and out of the critical section is determined by exponential distribution. The total runtime and time spent in the critical section is calculated. Sends completion message to parent process after the process completes k entries to critical section.

receive():
Receives messages from other processes until exit message is received. If a reply message is received, update the process variables and the willReceive list. Once reply is received, the sender will not receive the request again. If a request message is received, check the state of the current process. If the process is already in the critical section, defer the sender's request. If the process wants to enter the critical section, compare the lamport clocks of the processes. If the current process has higher priority, defer the sender's request. Otherwise, check if the sender is in the willReceive list. If not, send a request to that process. Then, send a reply to the sender. The request is sent to ensure that mutual exclusion is maintained. If the process is not in the critical section and does not want to enter the critical section, send a reply to the sender.

main():
Read the input parameters from the file. Creates n processes. The parent process sends exit messages to all processes and waits for all processes to complete. The child

processes start the threads to do the work and receive messages. The statistics are calculated and written to the file. The sockets are closed after the completion of the process.

Maekawa's Algorithm:

Network description:

- Each process has a socket to receive messages from other processes, called receiver
- Each process has a list of sockets to send messages to other processes, called senders
- Each process has a base endpoint, succeeded by the process id, to create a unique endpoint for each process

procVals class:

- quorum: list of processes in the quorum
- lockedOn: process on which the process is locked (first - lamport clock of the process, second - process id)
- queue: queue of processes requesting to enter CS
- numRepliesExpected: number of replies expected to enter CS
- numRepliesReceived: number of replies received
- numFailed: number of failed messages received
- numInquire: number of inquire messages received
- lamportClock: current lamport clock
- csEntryClock: clock at which process requested to enter CS
- eventMtx: mutex for process variables

reqCS(): request to enter critical section. Sets the variables, increment lamport clock, sets csEntryClock to current lamport clock

canEnterCS(): check if the process can enter the critical section. Possible only if number of replies received is equal to number of replies expected

init():

Initialize the quorum for each process, except the parent process. Also, initialize the sockets for each process.

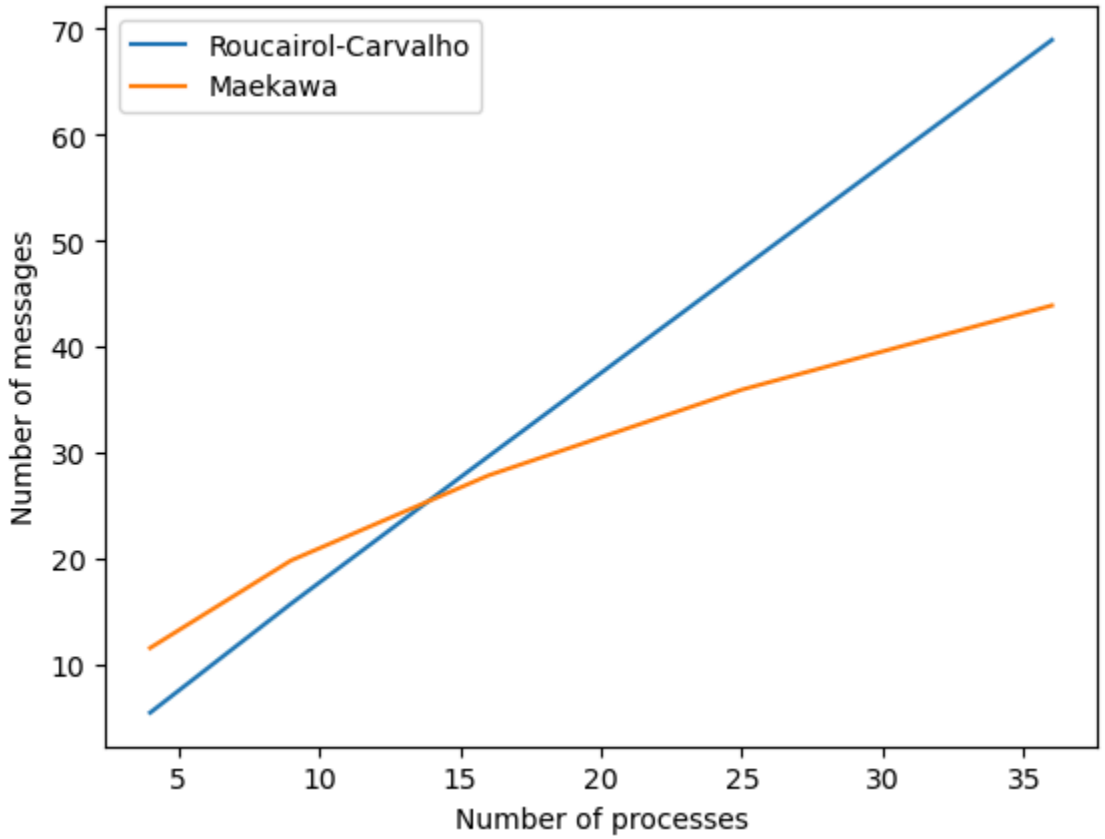
logEvent(string message):
Logs the event or sends the message to the parent process

receive():
Receives messages from other processes until exit message is received. If a reply message is received, update process variables. If a request message is received, add the request to the queue. If the lock has not been set yet, set the lock and send a reply to the process requesting to enter CS. If a lock has been set, check the priority of the process requesting to enter CS. If the process has lower priority, send a failed message. If the process has higher priority, send an inquire message to the process holding the lock. If a yield message is received and lock is held by the process sending the yield message, set the lock to the process with highest priority in the queue and send a reply to the process holding the lock. If a failed message is received and the process has previously received an inquire message, send a yield message to all processes in the quorum. If an inquire message is received and the process has previously received a failed message, send a yield message to all processes in the quorum. If a release message is received, remove the process from the queue. If the queue is not empty, send a reply to the process with highest priority in the queue. If the queue is empty, set the lock to null. If an exit message is received, break the loop.

main():
Read the input parameters from the file. Check if n is a perfect square. Create n processes. If the process id is n, run the loop until all processes have sent a termination message. Send exit messages to all processes. Wait for all processes to complete. Calculate the statistics. If the process id is not n, wait for all processes to be ready. Start the threads. Wait for the threads to complete. Close the log file. Close all sockets.

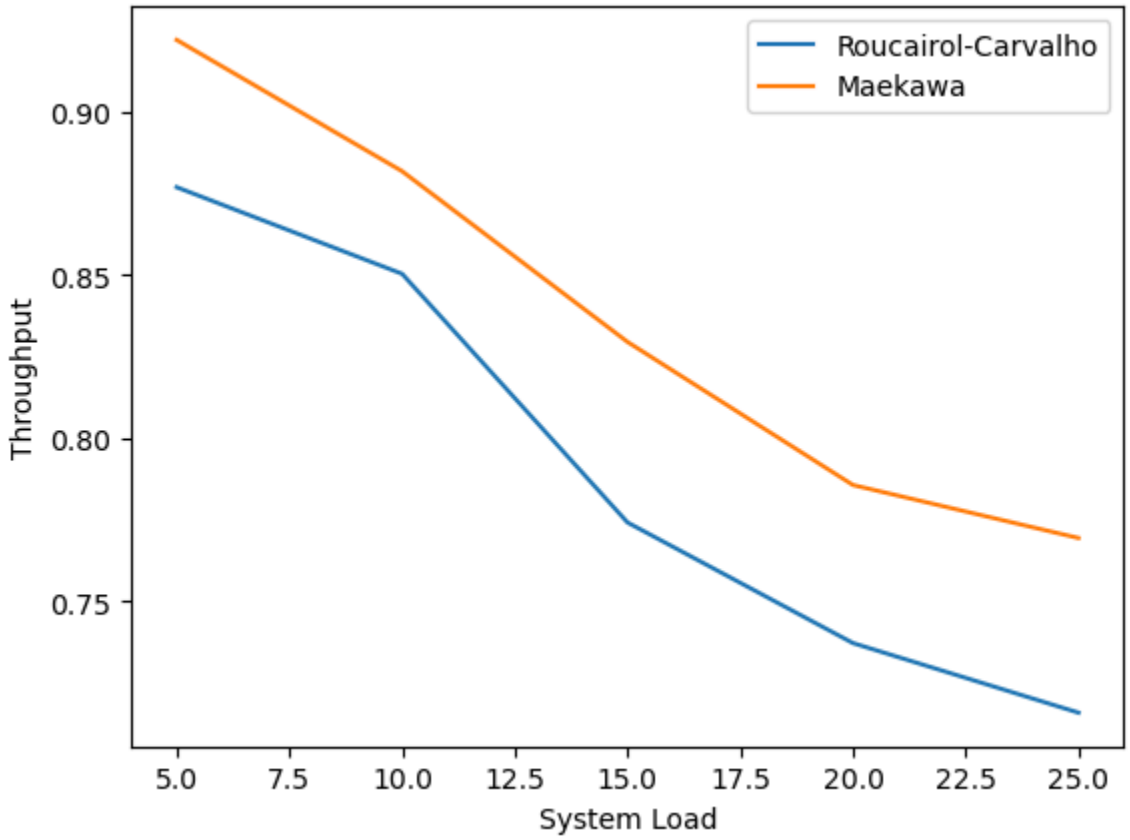
Results:

Number of processes vs number of messages (keeping number of CS entries, k = 15, alpha = 50, beta = 75):



1. For a smaller number of processes, Maekawa sends a higher number of messages compared to Roucairol-Carvalho since it has a large constant factor.
2. As the number of processes increase, the number of messages that Maekawa sends become smaller compared to Roucairol-Carvalho, since Maekawa has better message complexity of order of O(sqrt(N)) compared to O(N) for Roucairol-Carvalho.
3. The message complexity can also be seen in the graph, as Maekawa follows a curved line representing sqrt curve, while Roucairol-Carvalho follows a linear trend.

System load vs throughput (keeping number of processes, n = 9, alpha = 50, beta = 75):



1. For both Maekawa and Roucairol-Carvalho, throughput decreases as load increases.
2. Maekawa performs better than Roucairol-Carvalho in terms of throughput.