CS5320: Distributed Computing
Theory Assignment 2

Sagar Jain
CS17BTECH11034

March 25, 2020

Group Mutual Exclusion Algorithm Using a Message Passing System

The following Algorithm is one which converts the **decentralized solution proposed by Joung** to a message passing system instead of the shared memory model.

Assumptions

We shall assume there are two forums F and F'.
We assume that all the channels are FIFO (this is important since we will use other mutual exclusion algorithms as subroutines, and they require this to be true)

Approach

We will describe the algorithm with respect to a process i, which is trying to attend the forum F.
We can use **any message passing mutual exclusion algorithm** for our task, example **Lamport's Algorithm**. The following functions are exposed by Lamport's Algorithm:

- `enterCS(vals ...)`
- `releaseCS(vals ...)`

We shall use these two in our implementation, to access the resources which would have otherwise been in the shared memory.

The basic idea is to just convert all the instances of shared memory usage to message passing in Joung's algorithm.

The Algorithm

```
Define
flag: array[1..n   1] of (state, op) [Array which was earlier shared will be
    accessed through CS now]
turn ∈ {F, F‘} [Will be accessed through CS now]
state ∈ {request, in_cs, in_forum, passive}
op ∈ {F, F, N}
enterCS(flag)
do j != i: flag[j] = (in_cs, F′) →
   flag[i] := (request, F);   {request phase}
   releaseCS(flag)
   enterCS(flag, turn)
   do turn ≠ F ∧ ¬all_passive(F′) →
      releaseCS(flag, turn)
   od;
   if(haveResource(flag)) → releaseCS(flag)
      enterCS(flag)
   fi;
   flag[i] := (in_cs, F);     {in_cs phase}
   releaseCS(flag)
od;
if(haveResource(flag)) → releaseCS(flag) fi;
attend forum F                {in_forum phase}
enterCS(turn)
turn := F′;
releaseCS(turn)
enterCS(flag)
flag[i] := (passive, N)       {passive phase}
releaseCS(flag)
```

$$all\_passive(F') \equiv \forall j \neq i : flag[j] = (state, op) \implies op \neq F'$$

Points to Note

The fairness incase of processes is handled by using the same was as the seconf Joung's algorithm by having a leader i.e. successor variable in every process and giving direct entry to other processes.

Whenever we exit the CS we always exit it after the requisite instructions are executed ,so , no process can use the resources indefinitely. Other consideration like starvation, fairness, etc depend on the mutual exclusion algoirthm chosen and on the fact that processes don't fail after entering the CS.