CS5320: Distributed Computing, Spring 2020

Theory Assignment 2: Group mutual exclusion algo

SAI HARSHA KOTTAPALLI

CS17BTECH11036

First attempt with two forums F and F '

define flag: array[1..n – 1] of (state, op), turn ∈ {F, F '}

state ∈ {request, in_cs, in_forum, passive}

op ∈ {F, F ', ⊥}

timestamp (this stores lamport's timestamp)

turn // can only be changed by process in CS a part of shared resource (similar to how CS is implemented)

{Program for process i trying to attend forum F}

Procedure handle_req() // here internally a queue is maintained based on

// lamport timestamp

while True

if queue.empty()

do nothing

else

proc_id = queue.front()

queue.pop()

send(all_state) // to process proc_id. state comprises of flag and op and state

---

```
Procedure handle_turn() // here internally a queue is maintained based on
                        // lamport timestamp
while True
        if queue_flag.empty()
                do nothing
        else
                turn_recv = queue_flag.front()
                queue.pop()
                turn = turn_recv


Procedure main
proc_id_ctr = 1
pass = True, req = False
while(proc_id_ctr != num_of_proc + 1)
        if (proc_id_ctr == proc_id)
                continue
        req_flag(timestamp) // request proc_id_ctr for flag
        recv(all_state) // from process proc_id_ctr
        if all_state.flag_inc = (in_cs, F ') →
                req = True
        if  all_state.op = F'
                pass = False        // simulates all_passive (F')
        proc_id_ctr++
if req = true
```

```
        flag := (request, F)                                    {request phase}
while ¬pass
        pass = True
        for proc_id_ctr = 1 to num_of_proc
                if (proc_id_ctr == proc_id)
                                continue
                req_flag(timestamp)
                recv(all_state) // from process proc_id
                if  all_state.op = F'
                        pass = false        // simulates all_passive (F')
while  turn ≠ F
        do nothing // busy wait until turn becomes F

flag := (in_cs, F);                                             {in_cs phase}
attend forum F;                                                 {in_forum phase}
turn := F';
for proc_id_ctr = 1 to num_of_proc
        if (proc_id_ctr == proc_id)
                continue
        set_turn(turn, timestamp) // sends to proc_id_ctr and inturn calls
        handle_turn

flag := (passive, ⊥)                                            {passive phase}
```

Notes

- handle_req and handle_flag runs concurrently to the main program.
- Requests are served based on its timestamp. That is, internally it receives every request and inserts into a priority queue “queue” and “queue_flag” based on lamport timestamp. This makes sure there is no deadlock.
- The channels have to be in FIFO.
- “turn” can only be changed by process in CS and then broadcasted to everyone