Dijkstra's Mutual Exclusion Protocol

- Tushar, 16D100009

Introduction

- The protocol was introduced by Dijkstra in 1965 in the paper - Solution of a problem in concurrent programming control
- In the original paper, he refers to the individual processes as ‘computers’
- Dijkstra identified even in 1965 that though the entire setting of N ‘computers’ and the critical section might appear to be academic, anyone who works with ‘computer coupling’ will appreciate the solution
- Dijkstra also requested the readers to try and think of an algorithm without reading the protocol to see why the solution was far from trivial

Protocol

```
interested[i] ← true
done ← false
loop
        loop % Entry Phase 1
                exit when k = i
                if not interested[k] then k ← i %try to go next
        end loop
        passed[i] ← true % begin Entry Phase 2
        done ← true
        for j ← 1..N except i % check if anyone else has passed Phase 1
                if passed[j] then passed[i] ← false; done ← false %go back to Phase 1
        end for
        exit when done
end loop
CRITICAL SECTION
passed[i] ← false
interested[i] ← false
REMAINDER SECTION
```

Protocol

- Processes : 1..N
- Shared variables : interested[1:N], passed[1:N], k
- Local variable : done
- interested[1:N] stores which process(es) are interested in entering the critical section
- passed[1:N] indicates which process(es) have passed phase 1 of the protocol
- passed[1:N] is the main array which helps in implementing mutual exclusion
- k is the process which is allowed to try for the resource next
- done is the local variable which indicates that the process can enter the critical section

Protocol

```
interested[i] ← true
done ← false
loop
        loop % Entry Phase 1
                exit when k = i
                if not interested[k] then k ← i % try to go next
        end loop
        passed[i] ← true % begin Entry Phase 2
        done ← true
        for j ← 1..N except i % check if anyone else has passed Phase 1
                if passed[j] then passed[i] ← false; done ← false % go back to Phase 1
        end for
        exit when done
end loop
CRITICAL SECTION
passed[i] ← false
interested[i] ← false
REMAINDER SECTION
```

Proof of mutual exclusion

- No two processes are in the critical section at the same time
- We prove this by contradiction
- Suppose 2 processes i and j are in the critical section at the same time
- Consider the last time each of them set their entry of passed to true before entering the critical section
- Suppose i did it after j
- Then, when i reads passed[j] in its phase 2, it will see that passed[j] is TRUE and will go back to phase 1
- This contradicts the fact that i goes into the critical section

Protocol

```
interested[i] ← true
done ← false
loop
        loop % Entry Phase 1
                exit when k = i
                if not interested[k] then k ← i % try to go next
        end loop
        passed[i] ← true % begin Entry Phase 2
        done ← true
        for j ← 1..N except i % check if anyone else has passed Phase 1
                if passed[j] then passed[i] ← false; done ← false % go back to Phase 1
        end for
        exit when done
end loop
CRITICAL SECTION
passed[i] ← false
interested[i] ← false
REMAINDER SECTION
```

Proof of Progress

- Here, the statement is that if somebody wants to enter the critical section, eventually someone does get in
- Suppose some processes are trying to get in the critical section and there is no process currently in the critical section
- Eventually, k will get set to the index of one of the processes trying to enter
- The value of k may change a few times but eventually k will stop changing until some progress is made because interested[k] is TRUE
- Once it has settled down, that process will wait till other processes are kicked out from Phase 2 and cannot re-enter
- Then, k will eventually get into the critical section

Other notes on the protocol

- Thus, this protocol implements mutual exclusion and progress as a whole
- The individual threads however might starve - one process might be left out of the critical section while another enters infinitely often
- The original protocol in the paper used slightly different notations - arrays b and c
- The interested array is a negation of b and passed is a negation of c
- The protocol uses 2N additional bits and 2 variables : k (shared) and done (local)