Name: G.Nithish
Reg No: 19BCS0012
Course: Operating system
Date: 17/2/2020

~~Digital~~ ~~Asses~~ Digital Assignment

①

1.4 Let

P0

```
flag[2];
int turn=0
flag[i]=1;
while(flag[j])
{
  if(turn==j)
  { flag[i]=0;
    while(turn==j);
    flag[i]==1;
  }
}
turn=j;
flag[i]=0;
```

P1

```
flag[2];
int turn=0;
flag[j]=1;
while(flag[i])
{
  if(turn==i)
  {
    flag[j]=0;
    while(turn==i);
    flag[j]=1;
  }
}
turn=i;
flag[j]=0;
```

solution for critical section:-

*) Mutual exclusion, *) progress,

*) Bounded waiting

i) Mutual Exclusion:

In the above process Po & P1 if the process Po 'Po' is executing in its critical section then no other process 'P1' can be execution C.S. Like wise, if P1 is executing In its critical section, then po can't be executib in C.S.

ii) Progress:-

Case i:-

If 'po' is showing intrest it is executed 'n' times when 'P1' is not showing interest to excecute

P_0	P_1
✓	×
✓	×
✓	×
⋮	⋮

Case ii:-

If 'p1' is showing intrest it is executed n times when 'Po' is not showing intrest to execute.

P_0	P_1
×	✓
×	✓
×	✓
⋮	⋮

iii) Bounded waiting

A bound must exist on the no. of times that 'Po' is allowed to enter its critical section after 'P1' has made a request to enter its critical section and before that request is granted.

∴ All the 3 condition satisfies the solution for critical section.

---

2) Let.

```
P0
while(true)
{ state[i]=interested;
turn=j;
while(state[j]==interested && turn==j);
  <<critical section>>
  state[i]= not interested;
  << code outside critical section>>
}
```

* In the above code it will enter into while loop. then state[i] will be assigned as interested and turn as j.
* After that using while loop it will check the condition.

While(state[j]==interested && turn==j) (4)

$\underbrace{\text{state[j]==interested}}_{\text{true}}$ && $\underbrace{\text{turn==j}}_{\text{False}}$

*) If both are true the o/p is true. otherwise false.

*) the condition fails and enter into critical section.

*) Then state[i] is assigned as not interested

the above code satisfies all the conditions for critical section, as follows.

*) Mutual Exclusion:

The process P_0 is executing in its critical section, then no other process can be execute in its critical section.

Progress

If a process is showing interest to execute then the other process will not show interest to execut, and vice versa.

Bounded Waiting:-

A bound must exist on the no. of times that other process is allowed to enter its critical section after a process has made to request to enter its critical section and before that request is granted

---

(3) Process P_i:-

```
do
 {
  flag[i]=TRUE;
  turn=i;
  while(flag[j] && turn==j);
 < critical section>
  flag[i]= FAlse;
   <remainder section>
 } while(True);
```

Nothing is wrong in the following variation of Peterson's algorithm since it satisfies all the 3 conditio for Critical section, as follows.

(i) Mutual Exclusion:-

If the Process P_i is executing in its critical section then no other Process can be executing in their Critical section.

(ii) progress:-

If no process is under execution in its Critical section and these allow the process that wish to enter to its critical section, then the selection of the Processes that will enter into the critical section and next can't be Postponed indefinitely.

(iii) Bounded waiting:-

A bound must exist on the no. of times that other processes are allowed to ~~ot~~ enter their critical Section after a process has mad a request to enter its Critical Section and before that request is granted.