Lecture-12
B Du

Date 15/02/18
Page

Initially turn = i

do {	do {
while (turn != i)	while (turn != j)
critical section	critical s
turn = j;	turn = i;
remainder section;	remainder section;
} while (1);	} while (1);

* When turn == i, P_j is ready to enter its critical section but P_i cannot. Then turn becomes i, but P_i is not there to enter its critical section yet, so P_j is given another chance but now turn == i, so it also cannot progress. In this way progress is hindered.
* If there is single process, that can't run again & again if needed
* Property of mutual exclusion is maintained, but not progress.

Initially ~~turn = i~~ ~~flag [i] = f~~ flag [j] = F

do {	do {
flag[i] = true;	flag [j] = true;
while (flag [j]);	while (flag [i])
critical section	critical section.
flag[i] = false;	flag [j] = true;
remainder section;	remainder section;
} while (1);	} while (1);

* If ~~for~~ both P_i and P_j come one after another, then they will run in synchronized manner. But if both comes at once, then they will get stuck in unbounded waiting. Doesn't guarantee progress & bounded waiting.

Scanned by CamScanner

Peterson algorithm

for 2 processes

Initially turn = i   flag [i] = F   flag [j] = F

```
do {
    flag [i] = true;
    turn = j;
    while (flag [j] && turn == j);
    // critical section;
    flag [i] = false;
    // remainder section.
} while (1);
```

```
do {
    flag [j] = true;
    turn = i;
    while (flag [i] && turn = i);
    critical section
    flag [j] = false;
    remainder section
} while (1);
```

for multiple processes

Initially choosing [0] = F   number [0] = 0
[1] = F   number [1] = 0

Entry section

exit section

```
do {
    choosing [i] = true;
    number [i] = max(number [0], number [1], ....., number[n-1]) + 1;
    choosing [i] = false;
    for (j = 0; j < n; j++) {
        while (choosing [j]);
        while ((number [j] != 0) && (number [j], j) < (number [i], i));
    }
    critical section;
    number [i] = 0;
    remainder section;
} while (1);
```