Lecture-11
BDU

Date 19/02/18
Page

Process synchronization -
- Processes that don't run independently, i.e. need some shared resource / info, need co-operation from other process.
- Producer process - fills the buffer ; counter++ ;
- Consumer process - consumes the buffer ; counter--;

Producer
```
while (true)
{ while (counter == BufferSize)
    buffer [in] = next Produced;
    in = in + 1;
    counter++;
}
```

Consumer
```
while (true)
{ while (counter == 0)
    next Consumed = buffer [out]
    out = out + 1;
    counter--;
}
```

T_0: reg1 = counter
T_1: reg1 = reg1 + 1
T_2: reg2 = counter
T_3: reg2 = reg2 - 1
T_4: counter = reg1
T_5: counter = reg2 = 4 or 6

But the actual value should be 5. → Race condition

Inconsistency of data due to lack of synchronization as multiple processes work on same data.

Race condition: A situation that creates inconsistency in data due to access to same address space to two or more process simultaneously.

Critical section: Code portion of ~~the~~ process where they work on common variables update shared files, tables etc.
Here, counter is the critical section
Rest of the code - remainder section

entry section, exit section

To achieve Synchronization -
- Mutual Exclusion
- Progress
- Bounded waiting

Mutual exclusion - Guarantees that when the process is in the critical section, no other process enter that critical section.

Progress - If no process is executing in the critical section, if some processes want to enter the critical section, then we select only those processes which are not working in its remainder section.

Bounded waiting - If a process is waiting for too long to ~~get~~ enter its critical section, while other processes keep entering & exiting the critical section, this should not happen. The waiting time of a process should be limited.

mutual exclusion

Peterson Algorithm

Process P_i:

```
do {
    flag[i] = True;
    turn = j;
    while (flag[j] && turn == j);
    // Critical section
    flag[i] = False;
    // Remainder section
} while (true);
```

entry section

exit section

→ gets stuck in the loop while P_j finished its job.

turn indicates that process P_i is allowed to execute in its critical section.

flag indication P_i is ready to enter critical section.

Process P_j:

```
do {
    flag[j] = true;
    turn = i;
    while (flag[i] && turn == i);
    // Critical section
    flag[j] = false;
    // Remainder section
} while (true);
```

entry section

exit section

→ loop breaks & critical section entered