Multiprocessor Scheduling

Example (1): A dual processor system uses FCFS scheduling with a common ready queue. Both processors become idle when queue contain processes with process times of 2.4, 6 , 6.7, 7.5, and 9.7.

Find the average waiting time for the above processes and compare it with the case of single processor with double speed.

Multiprocessor Scheduling

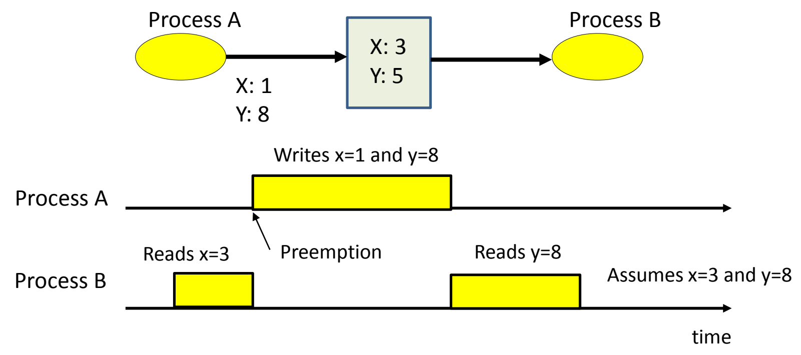
Example (2): A multitasking system runs four processes A, B, C, and D, with arrival times of 0, 2, 3 and 5 ms; and processing times of 3, 7, 4 and 8 ms respectively.

Calculate the average process waiting time assuming RR scheduling on two identical processors with q=2, with assignment of each new process to the least loaded processor.

Process Interaction and communication

Concurrent processes may interact in various ways.

Example:



Process Interaction and communication

Race condition: Multiple processes access and manipulate shared data with the outcome dependent on the relative timing of the processes.

Critical section: Part of process in which a shared data item is accessed.

Mutual exclusion requirement: The critical section of a process cannot be executed concurrently with the critical section of another process accessing the same data item.

How to enforce this mutual exclusion?

Mutual Exclusion

Using a shared binary flag to "lock" the data item

Pseudo-code for the critical section of each process is:

```
Shared int lock=0;
………
while (lock){};
lock =1;
Access_data_item();
lock =0;
```

Unless this flag can be test and set in one uninterruptible instruction, racing on flag itself can occur.

Mutual Exclusion

Alternation method

Assume that n processes access the shared data. Use a shared integer variable called turn.

Pseudo-code for the critical section of process i is:

```
while (turn !=i) { };
Access_data_item();
If (turn != n) turn =turn +1;
          else turn = 1;
```

- ❑ Number of processes should be known and fixed.
- ❑ A process may wait for processes in their non-critical sections.

Mutual Exclusion

Peterson's algorithm

The algorithm is given here for the case of two processes. The pseudo-code for process i (=1 or 2) is as follows:

```
.....
//non-critical section
.....
flag[i]= 1;
t= i;
while (flag[j]== true && t==i) { };
Access_data_item();
flag[i]= 0;
.....
```

Algorithm can be extended for the case of more than two processes.

Mutual Exclusion

we observe the following:

○ Solutions that allow the user to disable process switching are unacceptable. This capability can only be given to the system itself.

○ Solutions based on "busy waiting" waste CPU time. It is better to block the process that cannot enter a critical section.

○ It is desirable to have a solution which is independent of the number of processes.

Semaphores

A semaphore is a shared integer variable on which the following two *system* functions are defined

Down(s)

```
If s ≥ 1 then s:= s-1
else block the calling process
```

Up(s)

```
If there are processes blocked by down(s)
then unblock one of them
else s:= s+1;
```

down( ) and up( ) are executed *without interruption*.

Semaphores

We can enforce mutual exclusion using semaphores as follows: (m initially=1)

Process 1	Process 2	Process 3
……	……	……
<non-critical>	<non-critical>	<non-critical>
down(m);	**down**(m);	**down**(m);
critical section;	*critical section;*	*critical section;*
up(m);	**up**(m);	**up**(m);
<non-critical>	<non-critical>	<non-critical>
……	……	……