Name : Ananya Prasad
Reg No: 20BCE10093
Date : 16 NOV, 2021

Sub/Slot : CSE3003/B21+B22+B23
Exam : MIDSEM FALL 2021-22
Faculty : Dr. Abha Trivedi

1

Multiprocessor systems advantages :

* Economic : Most of the work is completed by the CPU, so these systems are economical. They share data storage, pheripheral devices and power supplies.
* Reliable : Unlike single processor systems, in multiprocessors if one processor fails, the system doesn't stop. They share their work between each other so that the work completes quickly.
* Energy efficient : In a single processor system there is more load as a lot of processes fill up; unlike those, multiprocessor systems execute these processes in a few time, so they consume less electricity.

Disadvantages of multiprocessor systems :

* Performance : The work is divided and is given to different processors if any processor fails to work. This takes more time to complete a work and performance gets affected.
* Not memory efficient : Alot of processors are working at the same time so each one needs its own memory space.
* Deadlock : If one process is using any input-output device, another processor at the same time cannot access that device which creates a deadlock.
* Expensive : They are cheaper in the long run, but expensive for the actual time.
* Higher level operating system required : As we are working with multiprocessors communication between the processors and the operating system becomes complicated.

So, this Justifies that multiprocessors have both, advantages and disadvantages.

(2)

Cache is useful when exchange of data has to take place and the components transferring are at different speeds. Cache solves the transfer problem by giving a 'buffer' of intermediate speeds suitable for both the components. The faster component does not need to wait for the slower component to transfer data if it finds the needed data in cache. The cache and the component, both have the same data now. If there is an update in the data now in the component, it should be updated into the cache as well. This prevents any issues in the multiprocessor systems where the data is used for different work by different processors.

A cache is made as large as a component, it may eliminate the ~~dev~~ component / device only when:

(i) The cache and component have same capacity to save the data when electricity is removed.

(ii) Cache is not expensive and should be affordable at par with the component

* In general, cache makes retrieving data from the memory more efficient. As it is more accessible to the processor, it increases the efficiency.

Operating system has Process scheduling queues. For all process states, there are different queues. for the same execution state, there are same queues. If the state changes, the process scheduling queue is removed from the current queue and moved to a new state queue.

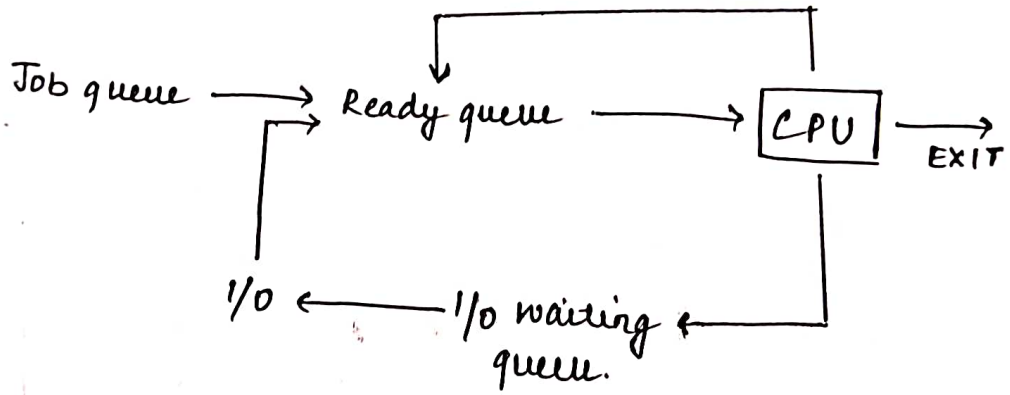
(i) Types of process scheduling queues:

(ii) Job queue: All processes in the system

(iii) Ready queue: For all the processes in main memory, about to be executed. (All new processes are pushed in this queue)

Device queue: If any I/O device is unavailable for use, it is pushed in this queue.

FUNC:



* Job queue: In the beginning, all the processes are stored here. It is lodged in the secondary memory. The job scheduler picks some jobs from here and put them in the primary memory for execution.

* Ready queue: It is maintained in the primary memory. The short ~~them~~ term Scheduler picks up the jobs and puts in the CPU for execution

* Waiting queue / device queue: If any process needs an I/O for execution, and the I/O is not empty, CPU makes this process to wait. It is stored in the waiting queue then and the process finishes when the I/O finishes its work.

4 (a) Preemptive SJF scheduling (SRTF) (4)

(Time at which the process gets the CPU for the first time − Arrival time)

Process	AT	BT	CT	TAT	WT (TAT−BT)	RT
P_1	5	10	22	17	7	7
P_2	3	5	8	5	0	0
P_3	0	18	37	37	19	0
P_4	6	4	12	6	2	2
Avg				65/4 = 16.25	28/4 = 7	9/4 = 2.25

Gantt chart:

P_3	P_2	P_4	P_1	P_3

0 3 8 12 22 37

(b) Preemptive Priority scheduling (lesser the number - higher the priority)

Priority	Process no	AT	BT	CT	TAT	WT	RT
3	P_1	5	10	24	19	9	9
1	P_2	3	5	8	5	0	0
4	P_3	0	5	26	26	21	0
2	P_4	4	6	14	10	4	4
Avg					60/4 = 15	34/4 = 8.5	13/4 = 3.25

Gantt chart:

P_3	P_2	P_4	P_1	P_3

0 3 8 14 24 26

(5)

Critical section protocols:

* Requirements:

1) Mutual Exclusion: If process P_i is executing in critical section, then no other process can execute in its critical section

(2) Progress: If no process is executing in its critical section and there exists some processes that wish to enter their CS, then the selection of the process that will enter the critical section next cannot be postponed indefinitely.

(3) Bound waiting: There exists a bound on the number of times that other processes are allowed to enter their critical section after a process has made a request to enter its CS and before that request is granted.

- Given in the question, initial values of flag are false.
- So, when the process enters the CS, it will turn slot to T. Let us consider that P_0 wants to enter the CS, flag[0] is turned to T. The while loop checks if P_1 wants to enter CS or not. Since, flag[1]=F, loop condition is false now and P_0 enters the CS.
- To check for mutual exclusion, let us consider that P_0 enter the CS at the time of context switched to P_1 and flag[1]=T. The loop checks and gives flag[0] to be T, so, P_1 is trapped in the while loop.
- Context switch occurs and we begin from P_0 again. After P_0 cross the CS, flag[0] is set to F. After another context switching, P_1 enters the CS as flag[0] is F. and the loop halts.
- Lets again check if P_0 can enter CS. flag[0] → T. The loop begins and gives flag[1] → T. so P_0 will be trapped in while loop. Mutual exclusion is satisfied.

P_0

flag [0] = T

while (flag[1]);

critical section

flag[0] = F

P_1

flag[1] = T

while (flag[0]);

critical section

flag[1] = F.

0	1
F	F

- For progress, flag is set to F. P_0 enters the CS and flag[0] is set to T. While loop begins and gives flag[1] → F, but we want T. Since P_1 is not entering the CS, P_0 again enters and comes out of the CS. And now flag[0] → F, again P_0 wants to enter CS and turns flag[0] → T The loop checks if flag[1] is T but P_1 is not entering so P_0 enters CS again. So it does not hold true for progress as it is not moving. and the system moves to deadlock.

For bound wait, the flag value keeps changing back to false.. No process here waits indefinitely and there is no bound on the number of times it can enter the CS.

So not all conditions are followed for the critical section protocols.

Thankyou Ma'am!