**Operating System - Assignment - I**

Q.1 Can a process ever go from Ready state to Block state? Can a process ever go from Blocked state to running state? Discuss your answer with reasoning.

Answer:

1. No, A process can never go from ready state to block state.

* Any process can be blocked only if it doesn’t require any CPU at that particular moment and waiting for some I/O task to be completed then the process will be blocked.
* Once I/O completes and process is ready, and need CPU for further processing then it will come back to ready queue.

1. No, Process cannot go from blocked state to running state.

* If process is blocked than it might be waiting for some input, which is not available yet. or It may be waiting for a completion of IO operation.
* Process is ready to run but operating system has decided to allocate CPU to some other process. When some external event or input for which process is waiting becomes available than process will go to ready state and if no other process is running at that time than it will go to running state.
* If CPU is allocated to some other process than current process has to wait for sometime in ready queue.

Q.2 What is the difference between user threads and systems threads?

Answer:

|  |  |
| --- | --- |
| User Level Threads | System Level Threads |
| Each process has its own Thread Control Block (TCB). | Thread table is maintained by the Kernel. |
| If one user level thread perform blocking operation then entire process will be blocked. | If one system thread perform blocking operation then another thread can continue execution. |
| Kernel is not aware about the number of threads in a single process and can’t make decisions based on number of threads. | Kernel is aware of system threads and can schedule processes based on number of threads in a single process. |
| User-level thread is generic and can run on any operating system. | System level thread is specific to the operating system. |
| Context switch time is less. | Context switch time is more. |
| Less overhead in terms of switching between threads. | More overhead in terms of switching between threads. |

Q. 3

|  |  |  |
| --- | --- | --- |
|  | Arrival Time | Burst Time |
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |
| P4 | 5 | 4 |

Timeline

1. P1 completion

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time | 0 | 2 | 4 | 5 | 7 |
|  | P1 arrives at 0 and starts running  (Time required by P1 : 7 sec) | P2 arrives but since P1 still running and not finished, P2 will wait in ready queue until it gets the chance.  (Time required by P2 : 4 sec) | P3 arrives but since P1 still running and not finished, P3 will wait in ready until it gets the chance.  (Time required by P3 : 1 sec) | P4 arrives but since P1 still running and not finished, P4 will wait in ready until it gets the chance.  (Time required by P4 : 4 sec) | P1 Finishes  and gets off CPU.  **Waiting time for P1: 0** |

1. In ready queue, there are 3 processes with time required as 4,1 and 4, So, the process which requires 1 sec (P3) will get the CPU.

|  |  |  |
| --- | --- | --- |
| Time | 7 | 8 |
|  | P3 gets the CPU and starts running.  (Time required by P3 : 1 sec)  (P2 and P4 waiting in the ready queue) | P3 finishes and gets off CPU  **Waiting time for P3: 7 - 4 , = 3 sec** |

1. In ready queue, P2 and P4 are waiting and each required 4 sec, but since P2 came first and as per queue norms, P2 will get CPU

|  |  |  |
| --- | --- | --- |
| Time | 8 | 12 |
|  | P2 gets the CPU and starts running.  (Time required by P2 : 4 sec) | P2 finishes and gets off CPU  **Waiting time for P2: 8 - 2 , = 6 sec** |

1. Only P4 is left in queue and gets the CPU

|  |  |  |
| --- | --- | --- |
| Time | 12 | 16 |
|  | P4 gets the CPU and starts running.  (Time required by P4 : 4 sec) | P4 finishes and gets off CPU  **Waiting time for P4: 12 - 5 , = 7 sec** |

Total waiting time: 0 + 3 + 6 + 7

= 16 sec;

Avg waiting time = 16/4

= 4 sec