**Operating System**

**Assignment 2**

**Aditya Parikh**

**ID: 121001**

**1.)**

* **Ready to run state:** When CPU cycles are available to a process then it is moved to run state.
* **Run to Ready state:** When time out occurs when process is already running then process is shifted to ready state.
* **Blocked to ready state:** When any process was already waiting for a resource and was temporarily blocked when it obtains those resource it is moved to ready state.
* **Ready to Non-resident state:** This event occurs if process is temporarily swapped out of main memory.
* **Ready to Blocked state:** If process is waiting for a resources which is not available then it is send to blocked state.
* **Blocked to Non-resident state:** If blocked queue is full and more process are to be blocked then those process which are blocked for longer time are moved to Non-resident state.

**2.)**

At time 22:

P1: I/O Blocked

P3: I/O Blocked

P5: Ready Running

P7: I/O Blocked

P8: Ready Running

At time 37:

P1: Ready Running

P3: Ready Running

P5: I/O Blocked

P7: I/O Blocked

P8: Ready Running

At time 47:

P1: Ready Running

P3: Ready Running

P5: Ready Suspended

P7: I/O Blocked

P8: Exit

**3.)** On success the process ID of the child process is returned to parent process and 0 is returned to child process.

**4.)** Threads at user level shared information about the process. So while switching between threads doesn’t require kernel level. So user level threads are fasted in terms of switching.

**5.)** Thread switching at ULT level does not require kernel context switching. While switching from process to process kernel should be notify about it and thus require kernel mode switching and which is a slower process. User level threads can run on any operating system and are bound to application level only and thus kernel awareness is not really required.

**6.)** While switching from process to process ULT’s have to move to kernel mode which is a slower process. While instead if the threads are created at kernel level the switching between processes can be faster. Another disadvantage of ULT is that when a system call is occurred, not only is that thread blocked, but also all of the threads within the process are blocked.

**7.)** When a system call is occurred while executing user level threads, not only the thread is blocked, but also all the threads within the process are blocked because those threads are internally depending upon each other within the process so if one thread gets blocked the others eventually gets blocked.

**8.)** When one to one mapping is done between kernel level threads and user level thread then when the system call occurs kernel level thread handles the system call which will speed up the process so multiple programs can run parallel which governs the concept of multithreaded programs.

**9.)** No if the process exists and there are still threads of that process running they will be not be able to continue.

**10.) Difference:**

|  |  |
| --- | --- |
| Competing Process | Cooperating Process |
| The process would compete for the resources. | The process would share the resources with some other process and at times even complete a task together with other processes. |
| Competing process is the process which does its work independent of any other process present. | Cooperating process is the one which does its work in accordance with the other present processes. |
| There is a careful isolation done among all the processes. | The processes are made to communicate and share with each other. |

**11.) Difference:**

|  |  |
| --- | --- |
| Strong Semaphore | Weak Semaphore |
| It specifies the order in which the processes should be removed from the waiting queue. | It does not specify the order from which the process should be removed from the waiting queue. |
| Mostly used by all the Operating System | Rarely used by any operating system |

1**2.)** Monitor is a synchronization algorithm that will block the process under same condition signal which will enter the monitor block in the sequence. When the process which was blocked under some condition occurs that process is resumed after the completion of the current running process.

**13.)** Blocking state is when either reader or writer or both are in blocked state after writing or before reading.

Non-blocking state is when either reader or writer or both are not blocked before receiving data or after writing data.

**14.)** No when there are less number of process which are waiting for the critical section access then busy waiting is efficient than blocked waiting because blocked waiting will require effort to store the process into the blocked queue which will take delay. While busy waiting will simple spin around the critical section until it gets free.

**15.)** They are equivalent in respect of functionality. Here s.count stands for number of process which can run simultaneously is calculated. When it turns to 0 it means that the further process should be blocked. The main difference is in the semSignal preemption. The semaphore determines if the process should be blocked depends on process queue instead of s.count.