OS Home Assignment

# 121027

1. RUNNING to READY: time slice overrun

READY to NONRESIDENT: Process swapped out, if memory limit reached

READY to RUNNING: dispatcher schedules a process by allotting a time slice

RUNNING to BLOCKED: a process issues an I/O or other kernel request.

BLOCKED to READY: if the awaited event completes (perhaps I/O completion)

BLOCKED to NONRESIDENT – Process swapped out, if memory limit reached

1. At time 22-

P1: Blocked for I/O

P3: Blocked for I/O

P5: Ready Running

P7: Blocked for I/O

P8: Ready Running

At time 37-

P1: Ready Running

P3: Ready Running

P5: Blocked for I/O

P7: Blocked for I/O

P8: Ready Running

At time 47-

P1: Ready Running

P3: Ready Running

P5: Ready Suspended

P7: Blocked for I/O

P8: Exit

1. 0
2. The reasons why a mode switch between threads is better than a process switch is-

a. The control blocks for processes are larger than for threads (hold more state information), so the amount of information to move during the thread switching is less than for process context switching.

b. The major reason is that the memory management is much simpler for threads than for processes. Threads share their memory so during mode switching, memory information does not have to be exchanged/changed, pages and page tables do not have to be switched, etc. This makes the thread context switch much cheaper than for processes. In case of processes the memory pieces (pages) need to be

exchanged, etc. (Will talk about the details in few weeks).

c. Threads do not have to worry about accounting, etc, so do not have to fill out all the information about accounting and other process specific information in their thread control block, so keeping the thread control block consistent is much faster .

d.Threads share files, so when mode switch happens in threads, these information stay the same and threads do not have to worry about it and that makes the mode switch much faster.



a. The process does not switch to the kernel mode to do thread management. This saves the overhead of two mode switches between user to kernel and kernel back to user.

b. Scheduling can be application specific. The scheduling algorithm can be changed to the application without disturbing the OS scheduler.

c. They are portable pieces of codes. No changes are required to the underlying kernel to support ULTs. The threads library is a set of application-level utilities shared by all applications.

1. When a ULT executes a system call, not only is that thread blocked, but also all of the threads within the process are blocked.

Here,a multithreaded application cannot take advantage of multiprocessing. A kernel assigns one process to only one processor at a time. Therefore, only a single thread within a process can execute at a time.

1. Switching between two threads is a job in user context which uses the usual resources. Whereas KLT are kernel context jobs and are rather viewed as independent processes. So, blocking of a KLT wouldn’t disturb the rest of the process but since ULTs aren’t kernel level they would block, blocking the process with them
2. If there is no one to one mapping between the ULTs of a process with KLTs, then, if any one ULT issues an I/O request for example, then no other ULT will be able to interact with the KLT until the I/O request is satisfied. This can hamper the progress rate of a process.
3. If a process exists then all the threads of that process will also stop running.
4. Competing Process

a. Competing process is the process which does its work independent of any other process present.

b. This process would compete for the resources.

c. There is a careful isolation done among all the processes.

Cooperating Process:

a. Cooperating process is the one which does its work in accordance with the other present processes.

b. This process would share the resources with some other process and at times even complete a task together with other processes.

c. The processes are made to communicate and share with each other.

11. Strong Semaphore:

a. It specifies the order in which the processes should be removed from the waiting queue.

b. Mostly used by all the Operating System

Weak Semaphore

a. It does not specify the order from which the process should be removed from the waiting queue.

b. Rarely used by any operating system

12. Monitor is a programming language construct that provides equivalent functionality to that of semaphores and is easier to control.

13. Blocking Send**:**   The process that receives the message and the one which sent the message both are blocked until the message is delivered

**Non-blocking Send: The process that sends the message is not blocked.**

14. No, busy waiting can be efficient if the expected wait time is shorter than the time it takes to pre-empt and re-schedule a thread. .

15. Yes, the functionality of both the code is same. But in the above given code, you cannot find the number of processes currently waiting in the queue. Every other functionality is same.