HOME ASSIGNMENT 2

VAISHALII UPADHYAY

121059

ANSWER 13

Everytime it gives a different value, as new iID is been creating

The program returns the process ID of the child process to the parent process.

ANSWER 4

The following is the reason why mode switch between the threads is better than the processes:

* the control blocks for processes are larger than for threads, so the amount of information to move during the thread switching is less than for process context switching
* 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.
* 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.

ANSWER 5

* Thread switching does not require kernel mode privileges because all of the thread management data structures are within the user address space of a single process. Therefore, the process does not switch to the kernel mode to do thread management. This saves the overhead of two mode switches
* Scheduling can be application specific. One application may benefit most from a simple round-robin scheduling algorithm, while another might benefit from a priority-based scheduling algorithm. The scheduling algorithm can be made according to the application without disturbing the underlying OS scheduler.
* ULTs can run on any OS. No changes are required to the underlying kernel to support ULTs.

ANSWER 6

* In a typical OS, many system calls are blocking. As a result, when a ULT executes a system call, not only is that thread blocked, but also all of the threads within the process are blocked.
* In a pure ULT strategy, 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. In effect, we have application-level multiprogramming within a single process. While this multiprogramming can result in a significant speedup of the application, there are applications that would benefit from the ability to execute portions of code simultaneously.

ANSWER 7

User process functions separately from Kernel processes. That is, thread structure of a process is not visible to the OS/kernel, which schedules on the basis of processes.

Hence once one thread is blocked, the whole process is blocked and consequently all threads in that process are blocked.

ANSWER 8

As in this environment we have one to one mapping, context switching will be easier. Though multi-progamming does not play a role in this system, but multi-threading will become faster as they do not have to wait for other threads.

ANSWER 9

No, once the process is terminated all the threads associated with that process will also terminate.

ANSWER 10

Competing processes - compete for resources. For example, two independent applications may both want to access the same disk or file or printer.

Cooperating Processes - Share resources. May or may not be aware of each other. Some processes are designed to cooperate together (jointly) on the same activity and share resources. They may also be aware of each other by process id.

ANSWER 11

Strong semaphore specifies in which order processes are removed from the waiting queue

Weak semaphore does not specify this.

ANSWER 12

The monitor is a programming-language construct that provides equivalent functionality to that of semaphores and that is easier to control.

ANSWER 13

blocking send or receive for messages means that the sender and/or reciever is blocked until the message is delivered. In a nonblocking send/recieve, neither the sender or reciever has to wait.

ANSWER 14

On average, yes, because busy-waiting consumes useless instruction cycles.

However, in a particular case, if a process comes to a point in the program where it

must wait for a condition to be satisfied, and if that condition is already satisfied,

then the busy-wait will find that out immediately, whereas, the blocking wait will

consume OS resources switching out of and back into the process

ANSWER 15

No, we cannot substitute one set for the other without altering the meaning of the program.

As the first program first checks the value of semaphore then updates it.

While second program updates the value of semaphore then checks it. This can give wrong output in terms of wait time.