

Shadow Homework 1 – System Programming (Processes and Threads)

Posted January 30, 2007 (NOT GRADED)

Problem 1: In a system with threads, is there normally one stack per thread or one stack per process? Explain.

There is one stack per thread since each thread needs its own variables to keep track of.

Problem 2: What are the three main states that a process can be in? Describe the meaning of each one briefly as well as the relation among them.

Three main states of a process are: (a) ready state when a process is ready to run on the CPU, (b) blocked state when a process is blocked waiting for a resource, information or event to happen (e.g., waiting for a mutex to be unlocked, (c) running state when a process holds the CPU and executes the individual program instructions.

Problem 3: What common events lead to the creation of a process?

- (a) need for multiprogramming to better utilize CPU-
- (b) need for concurrent execution of calculations during assistance of one active program to another
- (c) need for interactivity

Problem 4: What does it mean to preempt a process? Give an example when a running process must be preempted.

Preempting a process means that the dispatcher removes the process from the processor and put it either into the ready queue (if the time slice is up), or puts it into the I/O queue if the process was interrupted by an I/O event/system call.

An example when a running process must be preempted is, for example, when a specified time slice in time sharing OS expires. Then the process must be preempted. Another example would be if the process asks for a semaphore, but the semaphore is not free. Then the process must be preempted from the processor and wait in the semaphore queue for the semaphore to become free.

Problem 5: Justify why a process needs a blocked state. What kind of programs could processes run if we could have only running and ready states?

The process needs a blocking state if it runs system calls, where these calls need other services/OS routines to do the work on behalf of the process, and the process needs to wait for the OS routine results before it can continue its own work.

The programs that would survive only with running and ready states would be computational processes (and their programs) that do not need any I/O, are independent from each other, i.e., do not share any critical sections (variables), the memory for this process is pinned, i. e., the process does not run under virtual memory paradigm. Example would be to compute factorial of x.

Problem 6: Write a C program where a process creates a child process and each of them prints each other's PID, i.e, parent prints the child's PID and child prints the parent's PID.

See exercise 3.18, R&R

Problem 7: Consider the following steps when a process is switched: (a) Scheduler receives an interrupt; (b) Scheduler preempts the running process; (c) Scheduler selects another process for execution, (d) Scheduler updates the process control block of the selected process. Is this sequence of steps complete? Explain.

First of all, the answer will distinguish the roles of two entities scheduler and dispatcher. The scheduler will be the entity that works over the ready queue and selects a process according to the scheduling policy. Dispatcher entity will do the context switching.

The sequence of the steps is not quite complete for the following reasons: First, when a scheduler receives an interrupt, it means that this entity needs to select based on the scheduling policy which process to run from the ready queue. Once the selection of the process happens by the scheduler, the dispatcher needs to do mode and PCB switching, i.e., the currently running process needs to be preempted and have its PCB saved and moved to the ready queue, and new process' PCB needs to be loaded by the dispatcher to have the new process run on the processor. Each PCB needs to be correctly updated by the dispatcher.

Problem 8: List four reasons why a mode switch between threads may be cheaper than a mode switch between processes.

1. reason – 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
2. reason – 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).
3. reason – 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

4. reason – threads share files, so when mode switch happens in threads, these information stay the same and threads do not have to worry about it (similar to accounting information) and that makes the mode switch much faster.

But the most expensive operation is truly the memory aspect.

Problem 9: It was pointed out that two advantages of using multiple threads within a process are: (1) less work is involved in creating a new thread within an existing process than in creating a new process, and (2) communication among threads within the same process is simplified. Is it also the case that a mode switch between two threads within the same process involves less work than a mode switch between two threads in different processes?

Yes, it is the case. It means the mode switch between two threads within the same process is much cheaper as outlined in Problem 8. If two threads across two processes need to do mode switching, also process context switch has to happen and memory switch has to happen, etc. So it is much more expensive to switch two threads in two processes than two threads within the same process.

Problem 10: What resources are typically shared by all of the threads of a process?

Memory is shared; signal handlers are defined per process, so shared by all threads; files of the process are shared, accounting is shared (occurs only per process).