CS4348 : Operating Systems Concepts
Homework Assignment 2

Matthew McMillian
mgm160130@utdallas.edu

September 9, 2018

1. Explain how a process moves between the running, ready, and blocked states.

   When a process gets created it begins in a NEW state. When the OS is ready to accept the new process, it moves to the READY state, and eventually the RUNNING state. While running, the process can release itself and EXIT, timeout and move back to the READY state, or accept an event (an interrupt) and move to a BLOCKED state. Once the event (interrupt) has ended, the program goes back into a READY state until it is ready to begin RUNNING again.

2. There are three main sections of a Process Control Block. For each section, give an example of something it contains.

   The three main sections of a process control block are PROCESS IDENTIFICATION, PROCESS STATE INFORMATION, and PROCESS CONTROL INFORMATION, each one providing a different job for the process control block.

   - The PROCESS IDENTIFICATION section includes the numeric IDs for the process, the parent process, and the user of the process. This ID might be useful for cross-referencing other tables in later stages, or it could be used as an index.
   - The PROCESS STATE INFORMATION section includes user-visible registers, control and status registers, and stack pointers.
   - The PROCESS CONTROL INFORMATION sections is in charge of many tasks such as scheduling and state information (including process state, priority, etc.), data structuring the necessary pointers around the parent-child or queuing, privileges of the process for resources, manages the memory of the segment of memory, and interprocess communication.

3. Explain the steps of a process switch.

   A process switch occurs when the OS gains control from a current process (or in other words, whenever control of resources moves from a process back to the OS). This could happen due to an interrupt timeout, an exception, or a supervisor call. To swap modes, you must:

   - Save the process context (state).
   - Update the PCB of the current process to a non-running state.
   - Move the PCB of the process to an appropriate queue for the state.
   - Select another process to run (whichever the scheduler has next in queue).
   - Update the PCB of the selected process to a running state.
   - Updated the memory-management data structures of the new process.
   - Restore the state of the newly selected process to its previous context.

4. Why are multiple threads usually preferred to multiple processes?

   Multiple threads, in general, are better than multiple processes. Threads offer low creation time than spawning a new process and they take less time to terminate. On top of creation, threads are far faster at switching than processes and they can communicate far faster than processes can to each other. This is due to the fact that threads share the same memory, while processes occupy different parts of memory. IPC (Interprocess Communication) is known to be slow in comparison to a monolithic approach (in micro-kernel, micro-service, and multi-process approaches), so it makes sense that since all the memory is shared (connected) that the communication and swapping time is far faster.

5. Why do thread activities sometimes require coordinating?

   Imagine a toy scenario where you have two parts of a program we'll denote PART A and PART B. PART B depends on PART A for a certain piece of information. If we thread the process into two threads and we arrive PART B before we run PART A, we will be missing the value that we depend on. In general, thread coordination is used to make sure resources are managed properly when threads are being executed within a process (i.e. making sure that dependencies are met during process execution). Also if two threads are accessing the same resources of the program, data might get corrupted if the threads interfere with each other.

6. Which kind of threads can be scheduled across multiple processors? Explain why.

   Kernel-Level Threads can run across multiple processors. They are advantageous as they can schedule the same task across multiple processors for increased performance, so not only can a processes me multi-threaded but it can be spread across multiple processors! Kernel-Level Threads come from the design of the OS, and the OS schedules and manages these threads instead of entire processes. However, mode switching within these threads (especially different threads) can be very time consuming and it can significantly slow down a system.