Daniel Crawford

2018-09-08

Operating System Concepts Assignment 2

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

Whenever a process is first created, it is residing in the new state. That means that the processor is preparing to put it into memory, but it may not be completely ready to be put into virtual memory. This could be because there are too many processes already, and the system wants to set a limit, so performance will not be degraded. From the new state, it will decide if it can take on an additional process, and if so, will add it to the ready state. Now, the process will shift to the running state from the ready state. The dispatcher or the scheduler will handle when the processes run. If the program is ready to terminate, it will shift from the running state to the exit state. The only times this will happens is when the process indicates its completion, or it aborts the program. However, for some programs they will not be able to immediately complete their operations due to either execution time, I/O or interrupts. There are two cases for this, the first and most common one is that a process has entered the maximum allowable time for uninterrupted execution. Now, the process will go from the running state to the ready state, where it will be resumed once other processes have been executed. Another reason for this to happen is if in a operating systems which allocates priorities to processes, and one process holds a higher priority than another will cause this transition. When a program has a state which it requires some type of input or that it has to wait before execution will cause it to shift from a running state to a blocked state. Within this state, the process will be waiting for some necessary input so that it can continue its execution. Once this input is received, this process will shift back to a ready state. For the ready and exit states, the only way they can reach the exit state is if a parent process terminates them.

1. *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 the Process Control Block are the Process Identification, Processor state information and the Process control information. Starting with Process Identification, this is the way that a process can be distinguished from other processes. This is usually done using a unique numeric identifier. There are several uses of the identifier, such as cross-referencing process tables and indicating where a process is in each region. Also, when processes are communicating with each other, the process identifier tells the OS of the destination of the communication. Another example of what a process identifier contains is the information of parent processes and a user identifier. The process state information typically holds all the registers of a process. These registers need to be saved when a process gets interrupted so that a process can resume normally once it is running again. One of these registers inside the process state information is known as the program status word(PSW), which displays status information in the form of condition codes. All of these conditions code are flags and can represent identification flags, alignment checks, resume flags and many more since the PSW is 32 bit. This is an example of a status register within Program state information. The third section of a process control block is the Process control information. This section contains a lot of functionality for data structuring, interprocess communication, process privileges and more. The general information is the scheduling and state information provided. This can define the process state such as if its running, ready or blocked. A useful example of something contained in Process Control Information is scheduling-related information. This may tell the amount o time the process has been waiting, but it depends on the scheduling algorithm.

1. *Explain the steps of a process switch.*

The process switch starts at any moment in time when the OS gains control of a currently running process. There are three types of mechanisms for interrupting a process: Interrupts, Traps and Supervisor calls. A typical interrupt will be because of I/O, inner Clock or a Memory fault. A trap is due to an error or exception, in which case it must be handled, and determines if it can be fixed or has to be extinguished. Supervisor calls happen when an operating system level instruction is executed, so the process will transition to a routine that is part of the OS code. Now, after a mode switch, the steps for doing a process switch can begin. First, all the contents and registers in the processor need to be saved onto the stack. Secondly, the process control block needs to be changed from the running state and other metadata. Thirdly, the process control block needs to be moved to an appropriate queue. Next, another process needs to be designated for execution. Now the new process needs to have the process control block updated, such as changing the state to running from wherever it was before. Then the memory management data structures need to be updated but depends on how the address translation is managed. Finally, the context of the processor must be restored from when the process was last in its running state. Backtracking to mode switching, it is a step that is involved in the process switching steps, but it does not always involve process switching. It is the time spent in the interrupts process where it adjusts flags and looks for errors. What makes this not entirely part of the process switch is that it may not always switch a process, because it may return execution to the process it just interrupted.

1. *Why are multiple threads usually preferred to multiple processes?*

Within a single program, there are several reasons that threads will be preferred over separate processes. One reason is that threads can share information between process much more quickly because they do not need to call the kernel for interprocess communication. This is due to threads sharing the same memory and files, so communication is already quick since all the data is present in every thread. Threads also take far less time to create within an existing process than to make a entirely new process. This is thanks to all the threads sharing the same process control block and user address space, so each thread just needs to create a thread control block and user and kernel stack. An additional benefit is that threads can be terminated much more quickly than a process. Lastly, a thread can also perform switching between each other better than two processes can. So on the flipside with processes, if another process was made it would need load/reload all of the Process state information and control information. A thread maintains a lot of this information, so the setup phase for each thread is made much simpler and quicker.

1. Why do thread activities sometimes require coordinating?

Multithreading can be dangerous when managing a data structure because a data structure such as a binary search tree could be adjusted twice at one time by two threads. The issue with this is that the integrity of the data structure could be ruined, and there are some possible problems with destroying data. For this reason, synchronization is important in threads so that they do not interfere with each other or corrupt a data structure. The most dangerous aspect of using multiple threads is that they can change data and the other thread may be depending on that data. If threads are not coordinated with each other, they can cause major damage to important data.

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

These kinds of threads are called Kernel-Level Threads(KLT) and the mode must be switched to kernel mode to use this. The reason why a User-Level Thread(ULT) cannot be used across multiple processors is because it requires the kernel to assign the process to each processor. Therefore, a ULT will have threads but it will only run over one processor. In a KLT, each thread can be scheduled and provide this functionality. This is because threads can be treated as kernel-mode processes. An advantage is that if a thread is blocked, the kernel can schedule another thread for that same process. Another advantage is that multiple threads can be scheduled simultaneously. However, there is still a drawback. If one thread transfers control to another within the same process, it will require a mode switch to the kernel.