OPERATING SYSTEM

Lecture 5

Process Management: Threads

Dr Victoria Mukami

INTRODUCTION

This lecture is an introduction to process management and specifically threads. We will define various aspects regarding threads, identify the types of threads and review multithreading. We will review the various job and process states, and thread states. We will also review the multithreading models

Learning objectives

By the end of this topic, you should be able to:

1. Differentiate between threads and processes
2. Identify the types of threads, thread states and the role of multithreading
3. Understand the various multithreading models

OVERVIEW

We begin off with processor management and threads. During memory management lectures 2-4, we were dealing with one of the functions of an operating system: memory management. We reviewed how jobs are loaded into memory and how they are unloaded (read allocation and deallocation). Once a job is loaded into memory, then processing needs to occur. Processing is done by the CPU and involves several activities. Before looking at how scheduling is conducted (this will be done during the next lecture) we review some definitions. zsBefore, we can begin, let us review some definitions.

Processor: Also known as the CPU (Central Processing Unit). This is the main hardware that performs processing within a computer

Process: An active entity requiring resources to perform its function [1]. Also known as a task. A process is a single instance of a program

Program: An inactive entity that is composed of several processes [1]. This could be a stored file.

Thread: This is created by a process and can be scheduled and executed away from the main process [1]. A process can have many threads and owns the resources allocated to a thread. Not all operating systems support threads.

Multiprogramming: this refers to the ability of a processor to be allocated a job for a period and deallocated at the appropriate time. A single processor is shared by several jobs.

Multithreading: this is a feature that allows an operating system to manage processes with several threads [1]. One software that uses multithreading is the browser.

THREADS

All the threads of a process share resources of their process as seen above. They also reside in the same address space as the parent process and have access to the same data [2]. Sharing resources, the address space, and data means when one thread makes a change within a multithreading environment, the other threads will see the changes made. There are several benefits of threads regarding performance:

1. It is much faster to create a new thread in an existing process than it is to create a new process [2].
2. It takes less time to terminate a thread than a process [2].
3. Switching between two threads within the same process is much faster than switching between two processes.
4. Threads enhance communication between different executing programs [2].

Threads are implemented as either single or multi-threaded processes. One process can have a single thread or multiple threads as shown below.

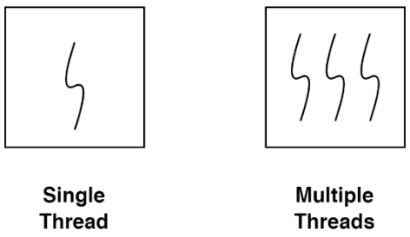

Figure 1: Single and Multi-threaded Process [2]

Process vs Threads

There are several differences between processors and threads, and they are highlighted below:

Table 1: Difference between Process and Thread

Process	Thread
Is resource intensive	Uses fewer resources than a thread
Takes a considerably longer time to terminate	Takes a shorter time to terminate
Takes a longer time to create	Takes a shorter time to the creation
Process switching must be done by the O/S	Thread switching does not involve the O/S
The process does not share data	Threads share data
Each process works independently of the other	Threads can read, write, or change each other's data

Thread States

A thread has different states as it moves through the system. When a process creates a thread, it is placed in the Ready queue [1]. When the thread is picked and it is assigned to a processor, it moves into the Running queue. When in the Running queue, a thread can move to one of three states: Waiting, Delayed or Blocked. When the thread is waiting for an external event then it moves to the Waiting state. Another reason why a thread can move to Waiting is when another higher priority process interrupts the current process. When complete a signal is sent to move from Waiting to Ready [1]. A thread can be delayed for a specific period and the thread moved to the Delayed state. When the time elapses, then the thread moves back to the Ready state. A thread is moved to the Blocked state when an I/O request is issued [1]. When the I/0 request is fulfilled then the thread moved to Ready. After a thread completes its transitions, it moves to the Finished state. The states are shown in an adapted Figure 2 [1].

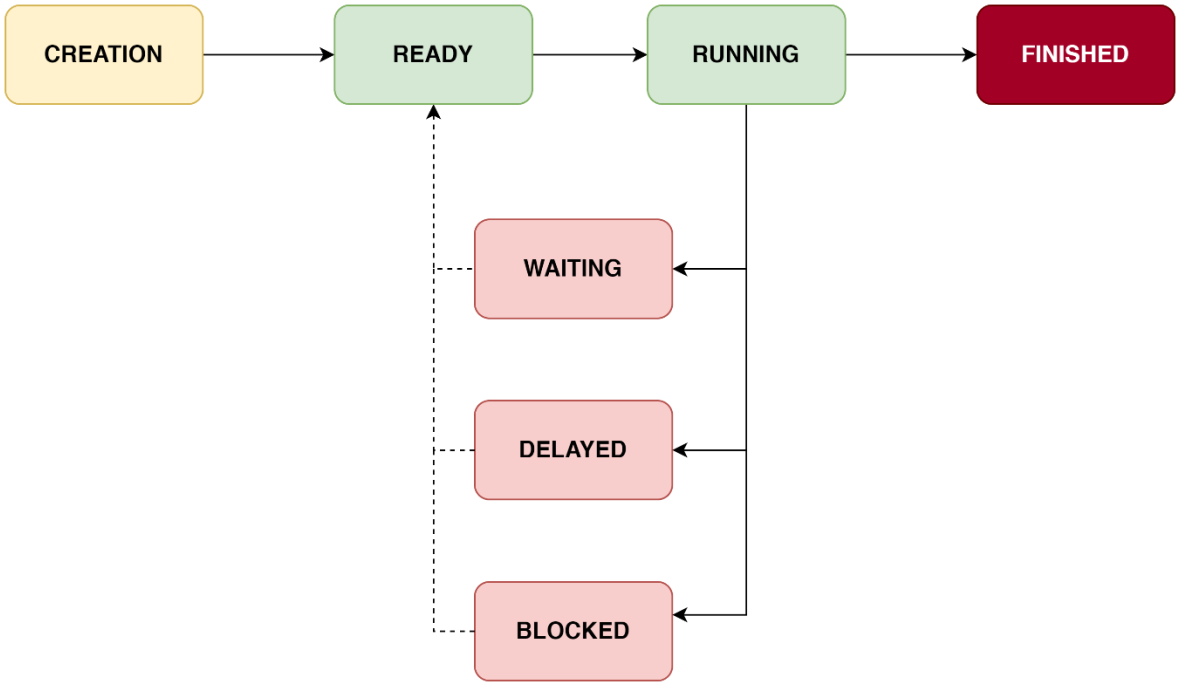

Figure 2:Thread states (Adapted) [1]

As shown, there are several states that a thread moves to. To be able to deal with this, the operating system must perform the following [1]:

1. Create new threads
2. Prepare a thread in readiness to execute
3. Delaying threads for a specific amount of time
4. Blocking threads that may have any I/O requests
5. Placing threads on wait if a specific event needs to take place or has occurred
6. Schedule threads for execution
7. Synchronizing thread execution
8. Terminate threads and release resources

Job and Process States

A job or process has various states as it moves through the system. After the job is accepted by the system, it is placed in the Hold state and placed in a queue. This could be a spool that has a table that has the various job characteristics including the CPU time required, memory required, and any special I/O devices needed [1]. Think of the Job Table from the memory management lecture. Through the job scheduler, the job moves to the Ready state. This will vary from system to system. Some systems will

place the job into the Ready queue directly without placing them on Hold. The job moves to the Running state when it is being executed. A Running job can move back to the Ready state. A job moves to the Waiting state if a particular resource is yet to be availed or if an I/0 request occurs. When these resources are availed then the process/job will move back to the Ready state. The Ready, Running and Waiting states are controlled by the processor scheduler. When the job completes then it moves to the Finished state. The Finished state like the Hold state is handled by the job scheduler. The various states are highlighted in Figure 3.

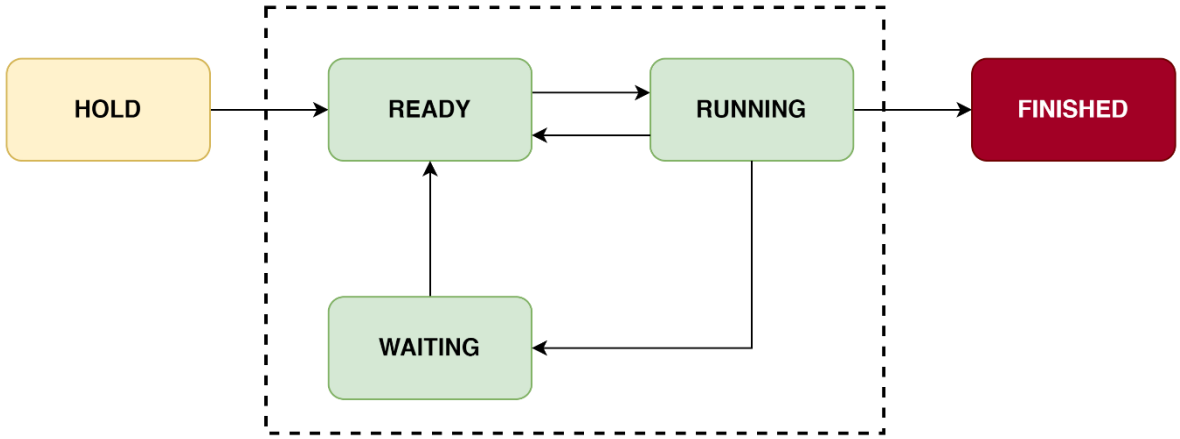

Figure 3: Job/Process States (Adapted) [1]

Types of Threads

Threads can be implemented as one of two kinds. The user-level threads and the kernel-level threads.

User-Level: The application or program manages the threads. The kernel is not aware of the existence of any threads. The thread library would contain code that would allow for the creation and termination of threads, passing data between threads and scheduling threads.

Kernel-Level: Thread management is conducted by the kernel. No thread management is done within the application/program. These threads are supported directly by the operating system. This is where an application can be programmed to be multithreaded. The kernel will maintain the information for the process as a whole and threads within the process.

MULTITHREADING

We already defined multithreading above. Operating systems sometimes provide for combined user-level and kernel-level threads facilities. There are three multithreading models.

Many-to-One: This model supports many user-level threads to one kernel-level thread. Thread management is done within the user's thread library.

One-to-One: This model has a one-to-one relationship between the user-level and the kernel-level threads. The model supports multiple threads to execute in parallel on microprocessors

Many-to-Many: This model supports multiple threads to an equal number or smaller number of kernel threads. The user-level threads and corresponding kernel-level threads can run in parallel on a multiprocessor machine.

SUMMARY

This lecture introduced process management and specifically threads. We defined various aspects regarding threads, identified the types of threads and reviewed multithreading. We also reviewed the various job and process states, and thread states. Finally, we reviewed the multithreading models.

DISCUSSION TOPIC

Conduct web research on one of the application programs running on your machine. Find out whether it supports threading and specifically multithreading. Gain an understanding of the type of multithreading model it supports. Present your findings in class.

REFERENCES

[1] McHoes, A., & Flynn, I., Understanding Operating Systems. Boston: Cengage Learning, 2018

[2] Stallings, W., Operating Systems: Internals and Design Principles. Harlow: Pearson Education Limited, 2018.