Grade: 60/60 A+

An Overview of Thread Implementation in Multi-Threaded Computer Systems

Alejandro Ricciardi

Colorado State University Global

CSC300: Operating Systems and Architecture

Joe Rangitsch

June 23, 2023

An Overview of Thread Implementation in Multi-Threaded Computer Systems

Threads are the fundamental component in multi-threaded computer systems which harvest the full potential of multi-core processors. Before defining a thread, it is important to understand what a process is. A process is a set of instructions that need to be executed by a processor and run by the OS scheduler service. A process consists of several elements such as stack (block of memory), code, data, program counter, and registers. A thread, on the other hand, is a sequence of instructions, within a process, that can be managed and run concurrently with other threads by the OS scheduler service. Threads are also called lightweight processes, they possess their own stack, program counter, and registers, but share code, data, and other operating system resources with other threads within the same process. (Silberschatz et al., 2018, p. 160). Furthermore, multi-threading can be defined as running multiple threads concurrently, or in parallel, and multicore computers can be defined as computer systems containing multiple processors or cores. In addition, several multi-threading models exist that utilize multicore computers, those models can be categorized as many-to-one, one-to-one, and many-to-many. Finally, multi-threaded computer systems bring to programs substantial benefits such as proficiency, responsiveness, resource sharing, memory economy, and scalability.

First, multi-threading is the ability of an OS to support multiple and concurrent threads within a process (Stallings, 2018, Chapter 4.1). Typically, a single-threaded system executes one threat by process, this approach can lead to potentially inefficient use of system resources, for example, the CPU remains idle when waiting for an I/O Operation. “MS-DOS is an example of an OS that supports a single-user process and a single thread” (Stallings, 2018, Chapter 4.1). On the other hand, multithreading, also described as task parallelism, allows multiple threads to be

executed concurrently on multiple cores, not to be confused with data parallelism which executes concurrently a single task on each multiple computing cores. Note that for an application to benefit from the multi-threaded computer systems, it needs to implement threading library functions using an application programming interface (API) (Neha, 2021). The implementation of multi-threading library functions within an application is generally referred to as parallel programming. In simple terms, parallel programming is the process of breaking down a problem into smaller tasks that can be executed at the same time using multiple computing resources (Burns, 2021). Also, multiple threads can be implemented on single-processor systems, depending on the thread priority and the OS policy installed on those machines, meaning that the threads are not executed concurrently. The term multi-threaded computer system is reserved for systems capable of running multiple threads simultaneously each on different cores.

Secondly, multicore computer systems contain multiple processors or cores, typically on a single chip. However, some systems may contain several multicore chips. Furthermore, multicore computer systems allow OS scheduler services capable of multi-threading to schedule multiple threads, within one process, to be executed concurrently each on different cores. Several multi-threading models exist, and they fall into one of the following categories, many-to-one, one-to-one, and many-to-many. The models are based on the concept of user-level threats and kernel-level threats.

In a many-to-one model, many user threads to one kernel thread, user threads are managed by the parent applications without the kernel being aware of the existence of threads (Stallings, 2018, Chapter 4.2). In other words, one thread within a process is executed at the time. Furthermore, the model approach does not harness multicore capabilities related to multi-threaded applications.

In a one-to-one model, one user thread to one kernel thread, the kernel is aware of the existence of threads and can manage the threads. It is one of the earliest implementations of what can be defined as multi-threading. Furthermore, by mapping each user thread to a kernel thread, the model provides more concurrency than the many-to-one model, and it also allows multiple threads to run in parallel on multiprocessors. The drawback of this model is that if a large number many kernel threads need to be generated to accommodate the user kernels, it could significantly affect the performance of the system (Silberschatz et al., 2018, p. 167).

In a many-to-many model, many level-user threads to many kernel-level threads, the user-level threads are mapped to an equal or smaller number of kernel-level threads. The model, also called the two-level model, creates the necessary number of kernel-level threads without overwhelming the system and limiting the number of user-level threads. The model can harvest fully the capacity of multi-threaded computer systems.

To summarize, threads are the fundamental component in multi-threaded computer systems. Multicore computer systems contain multiple processors or cores. Multi-threading, or task parallelism, is the ability of an OS to support multiple and concurrent threads, and multi-threaded computer systems are systems capable of running multiple threads simultaneously on several cores. Additionally, multicore computer systems contain multiple processors or cores, and they allow OS scheduler services capable of multi-threading to schedule multiple threads to be executed concurrently each on different cores. Furthermore, three multi-threading model categories exist, the many-to-one model, the one-to-one model, and the many-to-many model. Finally, multi-threaded computer systems bring to programs substantial benefits such as proficiency, responsiveness, resource sharing, memory economy, and scalability.

References

Burns, B. (2021, January 21). *What Is Parallel Programming?* TotalView by Perforce. https://totalview.io/blog/what-is-parallel-programming.

Neha, T. (2021, January 8). *What are thread libraries in os? pthread, win32, Java*. Binary Terms. https://binaryterms.com/thread-libraries-in-os.html.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* [PDF]. Wiley. Retrieved from: https://os.ecci.ucr.ac.cr/slides/Abraham-Silberschatz-Operating-System-Concepts-10th-2018.pdf

Stallings, W. (2018). Operating systems: Internals and design principles. Pearson.