International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 10, pp: 170-174 www.ijaem.net ISSN: 2395-5252

Organisation of Threads in Operating Systems

Karthik Raja K,Sujith D, Vignesh R, Dr.M.Sujithra M.C.A, M.Phil., PhD, Dr.A.D. Chitra M.C.A, M.Phil., PhD,

2nd Year, M.Sc. Software Systems (Integrated), Coimbatore Institute of Technology, Coimbatore
Assistant Professor, Department of Data Science, Coimbatore Institute of Technology, Coimbatore.
Assistant Professor, Department of Software Systems, Coimbatore Institute of Technology, Coimbatore.

Date of Submission: 20-11-2020 Date of Acceptance: 06-12-2020

ABSTRACT: The core functionality of Operating Systems (OS) is to keep flow of processes or execution of system running. The OS enforces flow of execution and smooth running of processes through implementing several hardware techniques, software applications and algorithms. This paper basically deals with threads used in an operating system. We have focused on the working and the ways of multithreaded system, how they are used to write a program in an efficient and effective way. This paper explains about what is an operating system andthe concepts of Threads in an operating system (OS).

KEYWORDS: Threads, multithread, effective, efficient, OS.

I. INTRODUCTION:

An operating system (OS) is software that controls computer hardware and software resources and provides common services for computer program. The operating system is an essential component of the software program in a computer system. Application programs usually require an operating system to function. For hardware functions such as input and output and memory allocation the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and will frequently make a system call to an OS function or be interrupted by it.

Examples of popular modern operating systems include Android, iOS, Linux, Microsoft windows and many more.

We shall now see how process management is done in OS to effectively keep the flow of the system with use of threads.

II. THREADS:

A thread is a single sequence stream within a process. They are sometimes called lightweight processes.In many perspective, threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel.

A thread consists of

- A program counter,
- A stack,
- A set of registers.

We use threads due to many reasons. Threads plays a key role in the designing of an operating system. A process with multiple threads makes a great server for example printer server. Because threads can share common data, they do not need to use inter process communication.Because of the very nature, threads can take advantage of multiprocessors.

2.1. Thread Libraries

- Thread libraries provide programmers with an API for creating and managing threads.
- Thread libraries may be implemented either in user space or in kernel space. The former involves API functions implemented solely within user space, with no kernel support. The latter involves system calls, and requires a kernel with thread library support.
- There are three main thread libraries in use today:

1. **POSIX-Pthreads** - may be provided as either a user or kernel library, as an extension to the POSIX standard.
2. **Win32 threads** - provided as a kernel-level library on Windows systems.
3. **Java threads** - Since Java generally runs on a Java Virtual Machine, the implementation of threads is based upon whatever OS and hardware the JVM is running on, i.e. either pthreads or Win32 threads depending on the system.

III. MULTI-THREADING

The ability of an OS to support multiple, concurrent paths of execution within a single process.Multithreading is mainly found in multitasking operating systems. Multithreading is a widespread programming and execution model that

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 10, pp: 170-174 www.ijaem.net ISSN: 2395-5252

allows multiple threads to exist within the context of a single process. These threads share the process's resources, but are able to execute independently. The threaded programming model provides developers with a useful abstraction of concurrent execution. Multithreading can also be applied to a single process to enable parallel execution on a multiprocessing system.

Most of the state information dealing with execution is maintained in thread-level data structures

- Suspending a process involves suspending all threads of the process
- Termination of a process terminates all threads within the process

Each thread has:

- An execution state (Running, Ready, Blocked)
- Saved thread context when not running
- An execution stack
- Some per-thread static storage for local variables
- Access to the memory and resources of its process (all threads of a process share this)

The key states for a thread are:

- **Running** - The process is active and running and being executed.
- **Ready** - The process is in a Ready state and waiting for its execution.
- **Blocked** - The process is in a state that cannot start its execution until some specific interruption occurs. That could be an I/O completion or any other interrupt.

3.1. Thread Synchronization

- It is necessary to synchronize the activities of the various threads
- All threads of a process share the same address space and other resources
- Any alteration of a resource by one thread affects the other threads in the same process

3.2. Benefits of Threads

- Takes less time to create and terminate a thread than a process
- Context switching are fast when working with threads.
- Enhance efficiency in communication between programs
- Threads are cheap as they only need a stack and storage for registers therefore, threads are cheap to create.

3.3. Single Threaded Approaches

A single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach. MS-DOS is an example

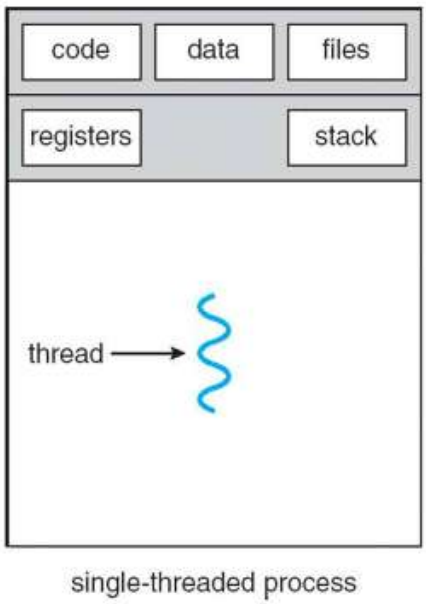

FIGURE 1: SINGLE THREADED PROCESS

3.4. Multi-Threaded Approaches

Multithreading - The ability of an OS to support multiple, concurrent paths of execution within a single process. These threads share the process's resources, but are able to execute independently.

Multithreading can also be applied to a single process to enable parallel execution on a multiprocessing system.A Java run-time environment is an example of a system of one process with multiple threads

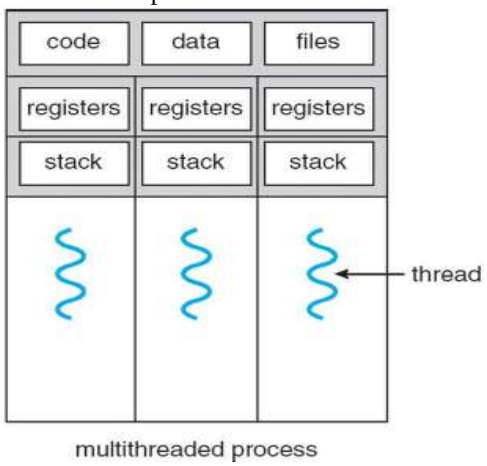

FIGURE 2: MULTI THREADED PROCESS

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 10, pp: 170-174 www.ijaem.net ISSN: 2395-5252

3.5. TYPES OF THREADS

Two types of threads

- User level threads
- Kernel level threads

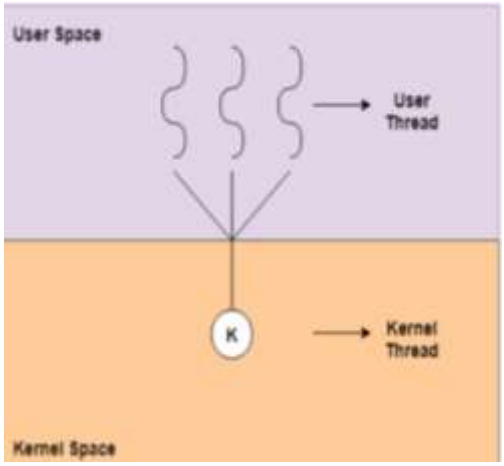

FIGURE 3: KERNEL AND USER SPACE

3.5.1. User-level threads

- Created and managed by a threads library that runs in the user space of a processwithout kernel support
- Only a single user-level thread within a process can execute at a time
- If one thread blocks, the entire process is blocked

Advantages of User-Level Threads

Some of the advantages of user-level threads are as follows −

- User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed.
- User-level threads can be run on any operating system.
- There are no kernel mode privileges required for thread switching in user-level threads.

Disadvantages of User-Level Threads

Some of the disadvantages of user-level threads are as follows −

- Multithreaded applications in user-level threads cannot use multiprocessing to their advantage.
- The entire process is blocked if one user-level thread performs blocking operation.

3.5.2. Kernel-level threads

- Kernel threads are supported within the kernel of the OS itself.
- Threads within a process that are maintained by the kernel
- Multiple threads within the same process can execute in parallel on a multiprocessor
- Blocking of a thread does not block the entire process

Advantages of Kernel-Level Threads

Some of the advantages of kernel-level threads are as follows −

- Multiple threads of the same process can be scheduled on different processors in kernel-level threads.
- The kernel routines can also be multithreaded.
- If a kernel-level thread is blocked, another thread of the same process can be scheduled by the kernel.

Disadvantages of Kernel-Level Threads

Some of the disadvantages of kernel-level threads are as follows −

- A mode switch to kernel mode is required to transfer control from one thread to another in a process.
- Kernel-level threads are slower to create as well as manage as compared to user-level threads.

IV. STRATEGIES OF MAPPING USER THREAD TO A KERNAL THREAD

In a specific implementation, the user threads must be mapped to kernel threads, using one of the following strategies.

4.1. Many to one

- ❖ Where many user level threads are mapped to a single kernel thread.
- ❖ In such model the thread is managed by thread library in the user space which makes such model efficient.
- ❖ However, the entire threads can be blocked if a thread makes a blocking system call.
- ❖ With multiprocessors system, threads can access the kernel with only a single thread at a time, multiple threads are unable to take advantage on multiprocessor and run in parallel on multiprocessors.
- ❖ With such model, developer will be able to create as many threads as required, however, the true concurrency is not implemented since the kernel can schedule only one thread at a time.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 2, Issue 10, pp: 170-174 www.ijaem.net ISSN: 2395-5252

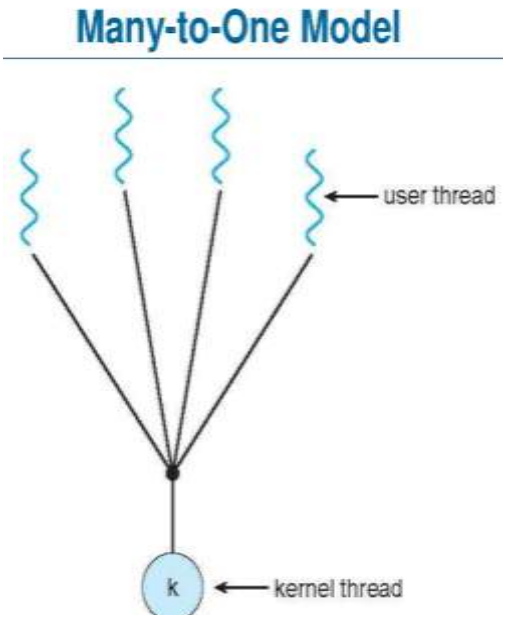

FIGURE 4: MANY TO ONE MODEL

4.2. One to one

- Where each user thread is mapped to a kernel thread.
- Such mechanism provides concurrency operation than the many-to–one model where other threads are allowed to even when one of these threads makes a blocking system call.
- Also, such system allows multiple threads to run in parallel on multiprocessors.
- Since such model creates a corresponding kernel thread with every creating of user thread, an overhead of creating kernel threads can affect the performance of an application and as such this implementation can be restricted with the number of threads that can be supported be the system.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.

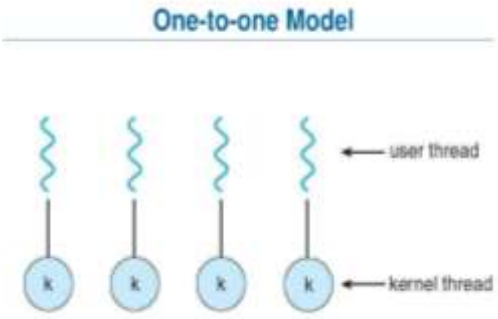

FIGURE 5: ONE TO ONE MODEL

4.3. Many to many

- The many to many modelmaps any number of user threads onto an equal or smaller, number of kernel threads, combining the best of the one-to –one and many to one models.
- Users have no restrictions on the number of threads created.
- Blocking kernel system calls do not block the entire process.
- Processed can be split across multiple process. Individual processes may be allocated variable number of kernel threads.

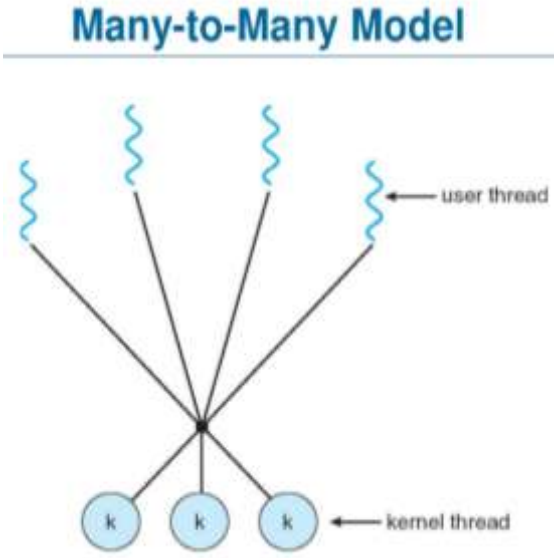

FIGURE 6: MANY TO MANY MODEL

TABLE 1:
STRATERGIES OF MAPPING USER THREAD TO KERNEL THREAD

	Restrictions on the number of threads	**Blocking kernel system calls**
Many to one	Users have no restrictions on the number of threads created.	Blocking a kernel system calls blocks the entire process.
One to one	Number of user threads can be restricted with the number of kernel threads that can be supported be the system.	Blocking kernel system calls do not block the entire process.
Many	Users have no	Blocking

to many	restrictions on the number of threads created.	kernel system calls do not block the entire process.

V. EXAMPLE OF THREADSINA WEB SERVER

Multiple web browsers (or browser window/tabs) connecting to the server at the same time should launch multiple threads in the server. Multiple threads allow multiple requests to be satisfied simultaneously, without having to service requests sequentially or to remove separate processes for every incoming request.Better Multi-Threaded Web Server that Handles More HTTP

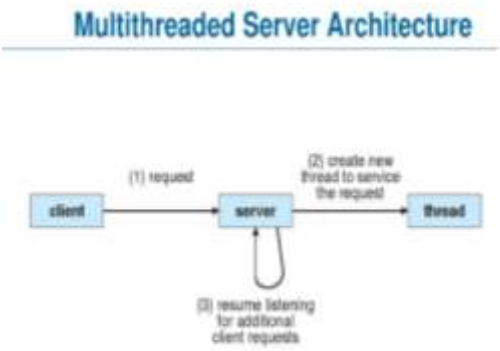

FIGURE 7: THREADING IN SERVER ARCHITECTURE

VI. CONCLUSION

A thread is a flow of control with a process and it is more efficient and more productive for a process to have multiple threads to achieve the maximum efficiency of any computing system (Titus, 2004). For example, with a server that can support multithreaded processes, such server can create several threads based on the client's requests. Multithreading allows application to be more interactive since the program can continue running even when part of such program's thread is blocked or is involved in a lengthy operation. Also, with multiprocessor architecture that is exist in modern computing system, different threads can run in parallel on different processors.

REFERENCES:

[1]. Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, "Operating System Concepts, Ninth Edition ", Chapter 4
[2]. https://home.ubalt.edu/abento/454/kernelprocesses/kernelprocesses.ppt
[3]. https://www.geeksforgeeks.org/thread-in-operating-system/
[4]. https://www.tutorialspoint.com/operating_system/os_multi_threading.htm