TECHNO INTERNATIONAL BATANAGAR

Subject: Operating Systen TERM PAPER(PCC-CS-502)

Topic: Threads in Operating System

Name: Ritam Sengupta

Department: Computer Science And Engineering

Roll No: 33200118019

Semester: 5th (3rd year)

Year: 2020

A TERM PAPER ON THREADS IN OPERATING SYSTEM:

INTRODUCTION:

There is a way of thread execution inside the process of any operating system. Apart from this, there can be more than one thread inside a process. Thread is often referred to as a lightweight process.

The process can be split down into so many threads. For example, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.

So in this term paper our main topic will be threads which are used in Operating System to control the flow of execution through the process code.

What is a Thread?

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Types of Threads:

In the operating system, there are two types of threads.

1. Kernel level thread.
2. User-level thread.

So lets shed some light on both types of threads;

User level Thread:

The operating system does not recognize the user-level thread. User threads can be easily implemented and it is implemented by the user. If a user performs a user-level thread blocking operation, the whole process is blocked. The kernel level thread does not know nothing about the user level thread. examples: Java thread, POSIX threads, etc.

Kernel Level Thread:

The kernel thread recognizes the operating system. There are a thread control block and process control block in the system for each thread and process in the kernel-level thread. The kernel-level thread is implemented by the operating system. The kernel knows about all the threads and manages them. The kernel-level thread offers a system call to create and manage the threads from user-space. The implementation of kernel threads is difficult than the user thread. Context switch time is longer in the kernel thread. If a kernel thread performs a blocking operation, the Banky thread execution can continue. Example: Window Solaris.

Advantages of User Level Threads:

- 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:

- 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.

Advantages of Kernel Level Threads:

1. Multiple threads of the same process can be scheduled on different processors in kernel-level threads.
2. The kernel routines can also be multithreaded.

3. If a kernel-level thread is blocked, another thread of the same process can be scheduled by the kernel.

Disadvantages of Kernel level Threads:

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

Components of Threads:

Any thread has the following components.

1. Program counter
2. Register set
3. Stack space

Why Multithreading?

The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Process vs Threads:

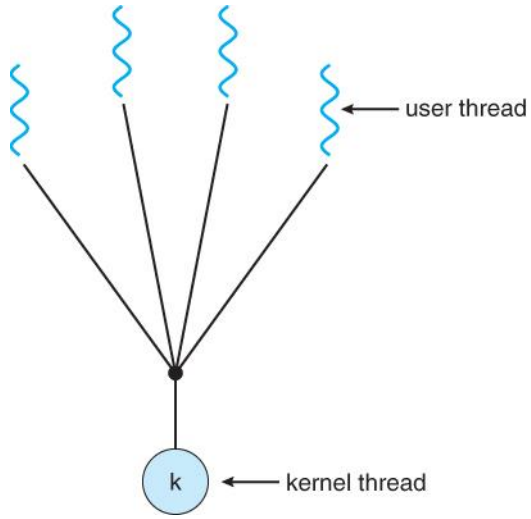
The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.

Multithreading Models:

1. There are two types of threads to be managed in a modern system: User threads and kernel threads.

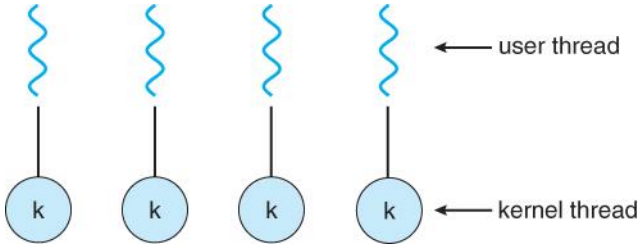
2. User threads are supported above the kernel, without kernel support. These are the threads that application programmers would put into their programs.
3. Kernel threads are supported within the kernel of the OS itself. All modern OSes support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.
4. In a specific implementation, the user threads must be mapped to kernel threads, using one of the following strategies.

Many to one Model:



Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

One to one Model:

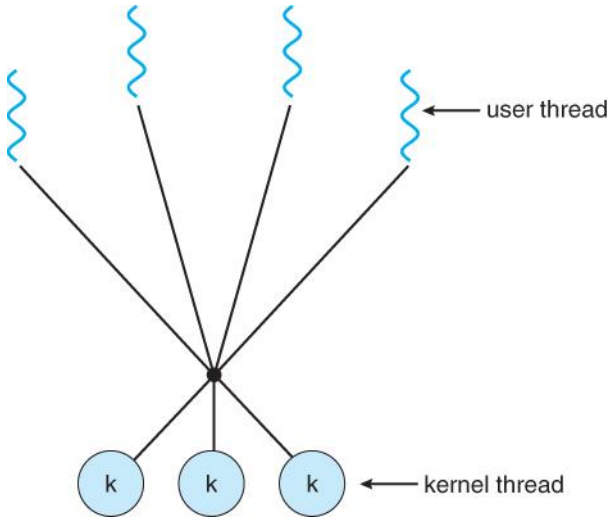


There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

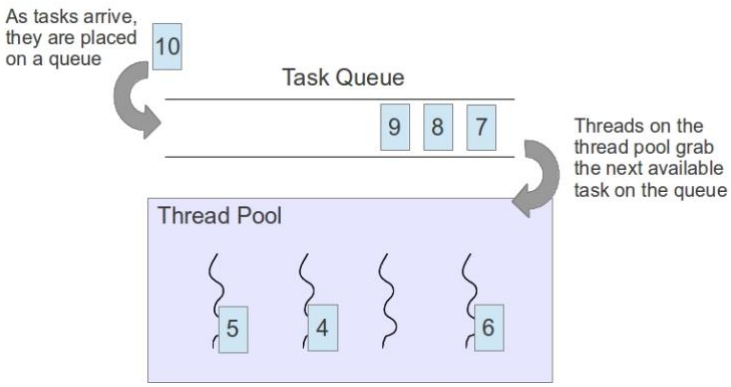
Many to many Model:

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.



Thread Pool:



In **Thread Pool** Instead of creating new threads when new tasks arrive, a **thread pool** keeps a number of idle threads that are ready for executing tasks as needed. After a thread completes execution of a task, it **does** not die. Instead it remains idle in the pool waiting to be chosen for executing new tasks. A **thread pool** is a collection of worker **threads** that efficiently execute asynchronous callbacks on behalf of the application. The **thread pool** is primarily **used** to reduce the number of application **threads** and provide management of the worker **threads**.

Advantages of Thread over Process:

1. **Responsiveness**: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. **Faster context switch:** Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

3**. Effective utilization of multiprocessor system:** If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. **Resource sharing:** Resources like code, data, and files can be shared among all threads within a process.
Note: stack and registers can't be shared among the threads. Each thread has its own stack and registers.

5. **Communication**: Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.

5. **Enhanced throughput of the system:** If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Multithreading Issues:

Thread Cancellation

Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is **Asynchronous cancellation**, which terminates the target thread immediately. The other is **Deferred cancellation** allows the target thread to periodically check if it should be cancelled.

Signal Handling

Signals are used in UNIX systems to notify a process that a particular event has occurred. Now in when a Multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all, or a single thread.

fork() System Call

fork() is a system call executed in the kernel through which a process creates a copy of itself. Now the problem in Multithreaded process is, if one thread forks, will the entire process be copied or not?

Security Issues

Yes, there can be security issues because of extensive sharing of resources between multiple threads.

----------THE. END----------