Threads

Thread:

1. Thread is the light-weight process or a component of process.
2. It is a path of execution within a process.
3. It is the smallest sequence of instructions that can be independently managed by a scheduler.
4. Multiple threads can exist within the same process and can share resources.
5. Each thread contains thread ID, a program counter, registers and a stack.
6. Context switching between threads in the same process is faster than the context switching between process.
7. Threads are not independent of each other as they share the data, code, resources etc. but can't share stack and registers.
8. Threads use same code and data so we might have to use critical section or locks.
9. Example: A web browser may have multiple threads like one thread to retrieve data from the network, one thread to display images on web page, one thread to render text on web page and so on.

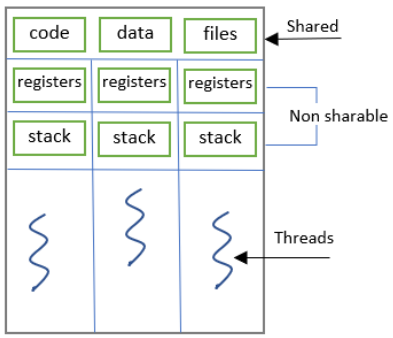
Similarities between Thread and Process:

1. Like Process, thread is active one at a time.
2. Both can create children.
3. Both executes sequentially.

Difference between Thread and Process:

Process	Thread
Processes are independent of each other.	Threads are not independent of each other as they exist as a part of a process.
Processes have separate address spaces.	Threads share their address space.
They have various states (ready, running, waiting etc.)	Multiple threads within a process share process state as well memory and resources.
System calls are involved.	No system calls.
Different copies of code and data	Same copies of code and data.

Multithreaded process:



Benefits of multithreaded programming:

1. Responsiveness: If multiple threads are in a process, if one thread executed completely then its output can be immediately available.
2. Fast context switching: Context switching between thread is less overhead to CPU.
3. Resource sharing: threads can share resources like code, data, memory.
4. Throughput.

Benefits of multithreading in detail: https://www.geeksforgeeks.org/operating-system-benefits-multithreading/

Introduction to thread in details: https://www.geeksforgeeks.org/operarting-system-thread/

Types of Thread:

There are two types of threads: User level thread and Kernel level thread.

A. User level thread:

1. User level thread is implemented in user level library.
2. User level thread can manage without kernel support.
3. User space contain all code and data structure for the library.

B. Kernel level thread:

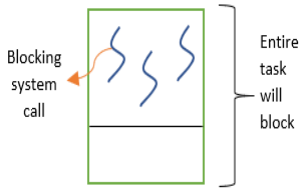
1. Kernel level threads are implemented by operating system.
2. Kernel level thread are managed directly by the operating system.
3. Kernel space contain all code and data structure for the library.

***Important links:**

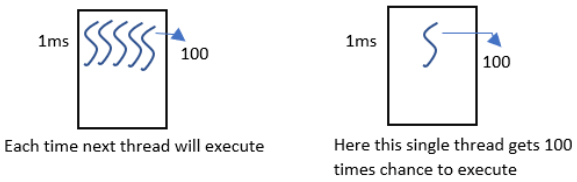
1. **Types of thread**: https://www.geeksforgeeks.org/operating-system-threads-types/
2. **User level thread VS kernel level thread**: https://www.geeksforgeeks.org/operating-system-user-level-thread-vs-kernel-level-thread/
3. **Thread based and Process based Multitasking**: https://www.geeksforgeeks.org/operating-system-process-based-thread-based-multitasking/

Problems with User level threads:

1. Blocking system call will block the task (if one thread makes block system call in user level threads then entire process will block).

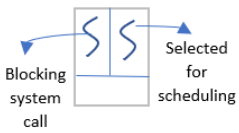


2. Unfair scheduling:



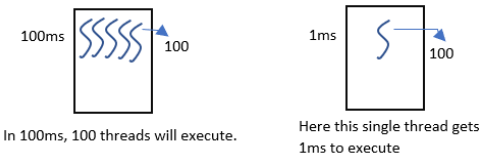
Problems of user level resolved by kernel level thread:

1. Blocking System Call:



If any thread makes block system call, then entire process will not be blocked as only that thread will block and the next thread will be scheduled.

2. Fair scheduling:



Problems in kernel level thread:

1. They are more expensive as compare to user level thread.
2. Switching in kernel level thread requires system call.

Multithreaded Models:

First read this link (explanations with the help of diagram): https://www.geeksforgeeks.org/multi-threading-model/

Summary:

1. **Many to One Model:** It maps many user-level threads to one kernel thread. In this, thread management is done by the thread library in user space. Whenever a thread makes blocking system call then entire process will block. Only one thread can access kernel at a time. In multiprocessor system, multiple threads are not able to run in parallel.
2. **One to One Model:** It maps each user-level thread to kernel level thread. It provides concurrency. In multiprocessor system, multiple threads can run in parallel.
3. **Many to Many Model**: It multiplexes many user-level threads to smaller or equal number of kernel threads.