CSC-203 Operating Systems[3]

Lecture: 6

Instructor: Sanjog Sigdel

sigdelsanjog.com.np

Date: May 22, 2019

Lecture 5

Unit2: Process Management

Threads: Definition, Thread vs Process, Thread Usage, User and Kernel Space Threads

Why would anyone want to have a kind of process within a process?

It turns out there are several reasons for having these mini processes, called threads. The main reason for having threads is that in many applications, **multiple activities are going on at once**. Some of these **may block from time to time**. By **decomposing** such an application into **multiple sequential threads** that run in **quasi-parallel**, the programming model becomes simpler.

Process Management

Threads:

Process are used to group resources together and threads are the entities scheduled for execution on the CPU

- Thread is a single sequence stream within in a process.
- In a process, threads allow multiple executions of streams
- Thread enable parallelism(The CPU Switches rapidly back and forth among the threads giving illusion that the threads are running in parallel)
- A thread can be in any of several states:
  - Running, Blocked, Ready or terminated
- Thread has its own stack(thread will generally call different procedures and thus a different execution history)

Process Management

Threads:

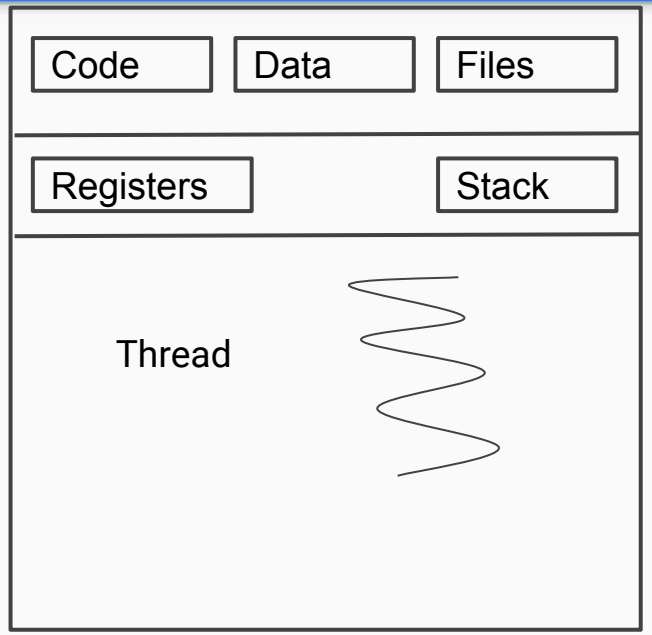
- Thread consists of a program counter, a register set and a stack space
- Threads are not independent of one other like processes
  - as a result threads share with other threads their code section data section,

Traditional processes have a single thread of control

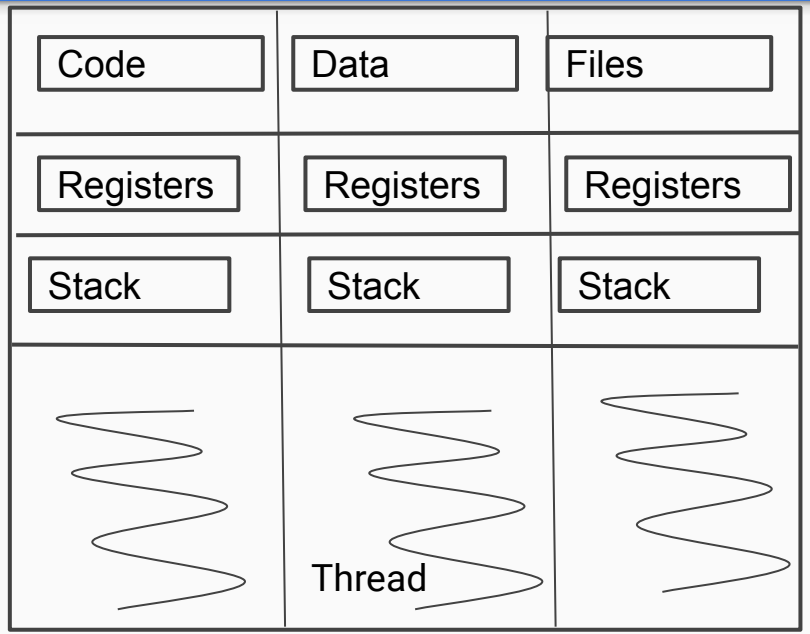
- One program counter and one sequence of instructions that can be carried out at any given time

Multiple threads within a single process, each having their own program counter, stack and set of registers, but sharing common code data, data and certain structures as open files

Process Management



Single THreaded Process



Multi THreaded Process

Multiple threads Example:

- In a word processor a background thread may check spelling and grammar while a foreground thread processes user input(keystrokes), while a third thread loads images from the hard drive and a fourth does periodic automatic backups of the file being edited.

Four Major Benefits of multi-threading

- **Responsiveness:** One thread may provide rapid response while the other threads are blocked or slow down doing intensive calculations
- **Resources sharing:** By default threads share common code, data, and other resources, which allow multiple tasks to be performed simultaneously in a single address space
- **Economy:** Creating and managing threads is much faster than performing the same task for processes
- **Scalability, i.e. Utilization of multiprocessor architectures:** execution of a multi-threaded application may be split amongst available processors

Properties of a Thread

- Only one system call can create more than one thread(Lightweight process)
- THreads share data and information
- THreads share instruction, global and heap regions but have its own individual stack and registers
- Thread management consumes no or fewer system calls as the communication between threads can be achieved using shared memory

Thread Usage

- First method is to put the threads package entirely in user space. Kernel knows nothing about them. Kernel is concerned in managing ordinary, single threaded processes.
- Thread run on top of a runtime system, which is a collection of procedures that manage threads
- When threads are managed in user space each process need its own private thread tables
- It is analogous to kernel's process table, except that it keeps track only of the pre-thread properties such each thread's program counter stack pointer, registers, state, etc.

Thread Usage

- Thread table is managed by runtime system
- When a thread is moved to ready state or blocked state, the information needed to restart it is stored in the thread table, exactly the same way as the kernel stores information about processes in the process table.

The Thread Model

There are two types of thread: User threads & Kernel threads

- **User threads** are supported above the kernel without kernel support
- These are the threads that application programmers would put into their programs
- **Kernel threads** are supported within the kernel of the OS itself
- All modern operating systems support kernel level threads, allowing the kernel to perform multiple simultaneous task.

Process Management

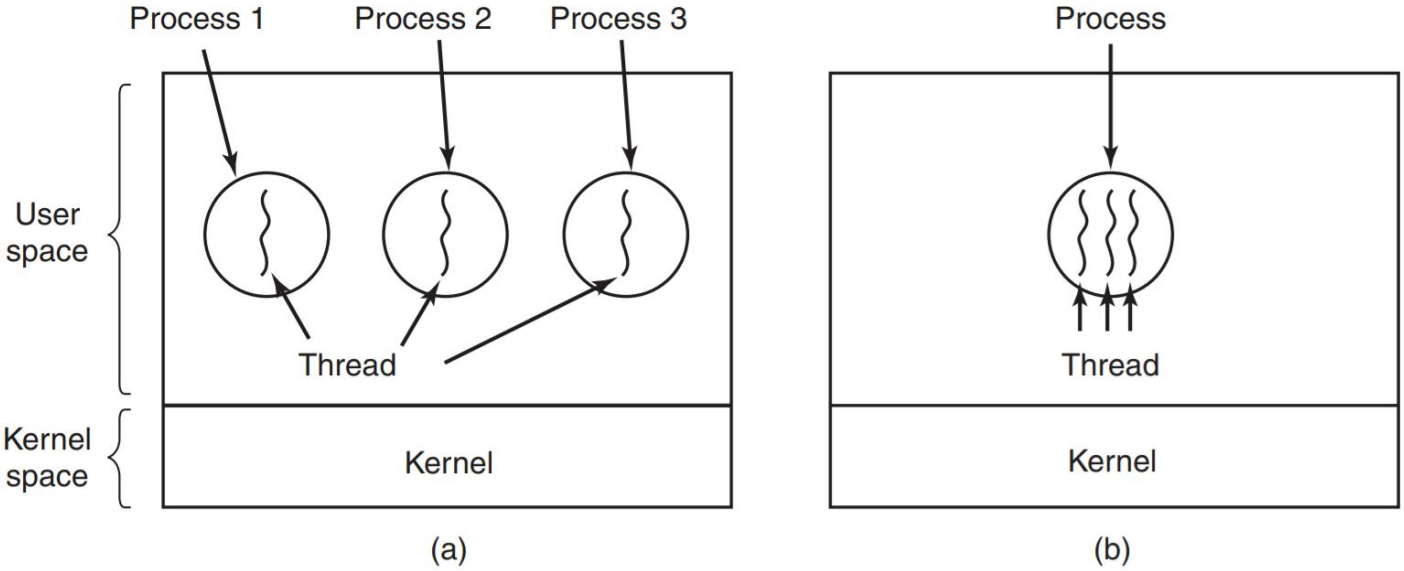
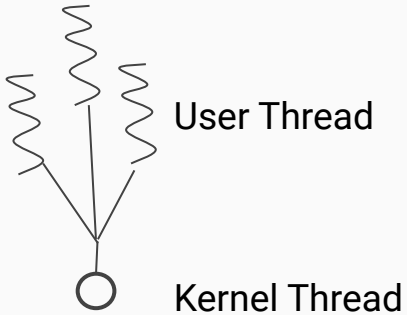


Figure 2-11. (a) Three processes each with one thread. (b) One process with three threads.

The Thread Model

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

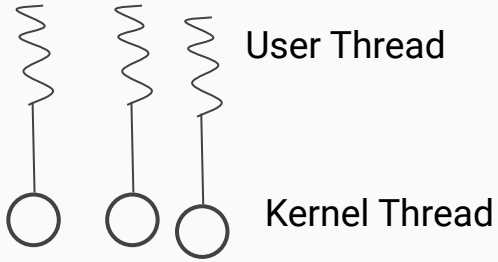
- Many To-One Model
  - Maps many user level threads to one kernel level thread. Thread management is done by thread library in user space.



Process Management

The Thread Model

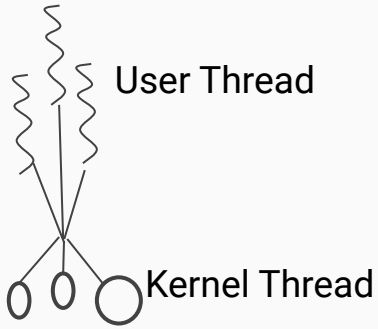
- One-To-One Model
  - One user thread is mapped to one kernel thread.
  - Provides more concurrency



Process Management

The Thread Model

- Many-To-Many Model
  - Many user level threads multiplex to the Kernel thread of smaller or equal numbers
  - Users have no restrictions on the number of threads created



Thread vs. Process

Similarities

- Like processes threads share CPU and only one thread active(running) at a time.
- Like processes threads within a process, threads within a process execute sequentially
- Like processes, thread can create children
- And like process, if one thread is blocked, another thread can run

Thread vs. Process

Differences

- All threads of a program are logically contained within a process
- A process is heavy weighted, butt a thread is light weighted
- A process is an isolated execution unit whereas thread is not isolated and shares memory
- A thread cannot have an individual existence; it is attached to a process. On the other hand, a process can exist individually.
- At the time of expiration of a thread, its associated stack could be recovered as every thread has its own stack. It contrast, if a process dies, all threads die including the process

Kernel Level Threads

- To make concurrency cheaper, the execution aspect of process is separated out into threads. All thread operations are implemented in kernel and the OS schedules all threads in the system.
- OS managed threads are called kernel-level threads or light weight processes.
- In Kernel Mode, kernel knows about and manages the threads
- Instead of thread table in each process, the kernel has a thread that keeps track of all threads in the system

Kernel Level Threads

Advantages

- Because kernel has full knowledge of all threads, Scheduler may decide to give more time to a process having large number of threads than process having small number of threads
- Kernel-level threads are especially good for applications that frequently block

Disadvantages

- Kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads
- Since kernel must manage and schedule threads as well as processes. It requires a full thread control block for each thread to maintain information about threads. As a result there is significant overhead and increased in kernel complexity

User-Level Threads

- To make threads cheap and fast, hey need to be implemented at user level. User-level threads are managed entirely by the run-time system(user-level library).
- User-level threads are small and fast.
- Creating a new thread, switching between threads and synchronizing threads are done via procedure call i.e. no kernel involvement.
- User-level threads are hundred times faster than Kernel-Level threads

User-Level Threads

Advantages

- User-level threads don't require modification to OS
- Simple representation: Each thread is represented simply by a Program Counter, registers, stack and a small control block, all stored in the user process address space
- SImple management: creating a thread, switching between threads and synchronization between threads can all be done without intervention of the kernel
- Fast and Efficient: Thread switching is not much more expensive than a procedure call

User-Level Threads

Disadvantages

- Lack of coordination between threads and OS kernel. Therefore process as whole gets one time slice irrespective of whether process has one thread a 1000 threads within.
- Require non-blocking system call i.e. multithreaded kernel. Otherwise, entire process will have blocked in the kernel, even if there are runnable threads left in the process.
    - If one thread causes a page fault, the process blocks

Next Lecture

Inter Process Communication: Definition, Race Condition, Critical Section

Submit Lab Report 1 by 25th May i.e. Saturday

Q&A

For Further Queries:
sigdelsanjog@gmail.com

Add [NISTBanepa] in Subject for any of your queries

THANK YOU