CS 343 - Operating Systems

Module-2E

Introduction to Threads

Dr. John Jose

Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati

Session Outline

- Process vs Threads
- Thread model
- Multithreaded programs
- User and Kernel threads
- Multithread mapping models
- Thread libraries and operations

Concept of Threads

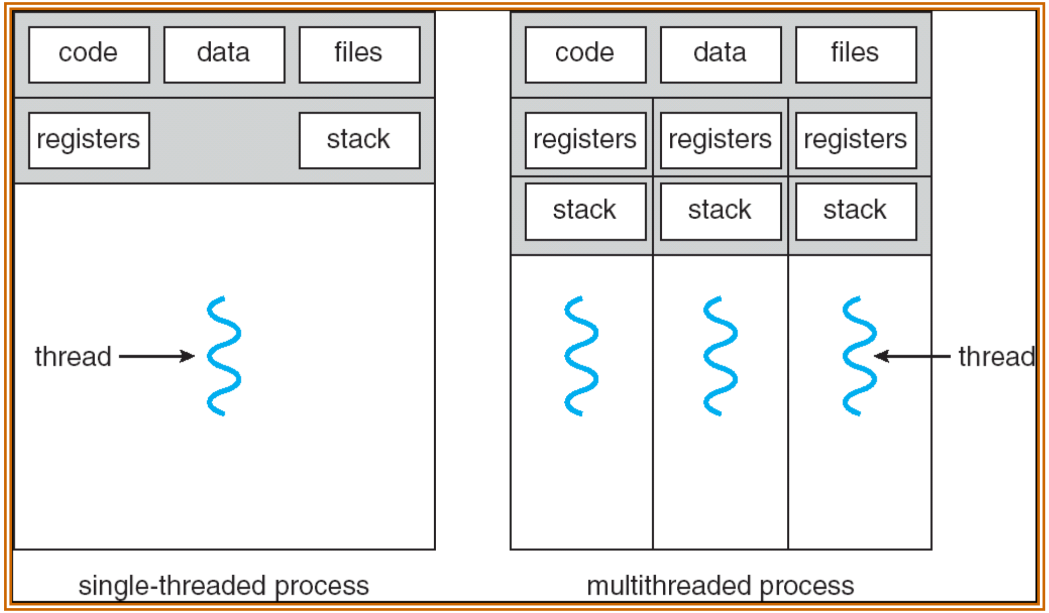
- Thread is a flow of control within a process.
  - single-threaded process, multi-threaded process.
- It is a basic unit of CPU utilization, which comprise
  - a thread ID, program counter, register set, stack.
- Shares with other threads belonging to the same process its code section, data section, and other OS resources (open files and signal)
- If a process has multiple threads of control, it can perform more than one task at a time.

The Thread Model

- Items shared by all threads in a process
- Items private to each thread

Per process items
Address space
Global variables
Open files
Child processes
Pending alarms
Signals and signal handlers
Accounting information

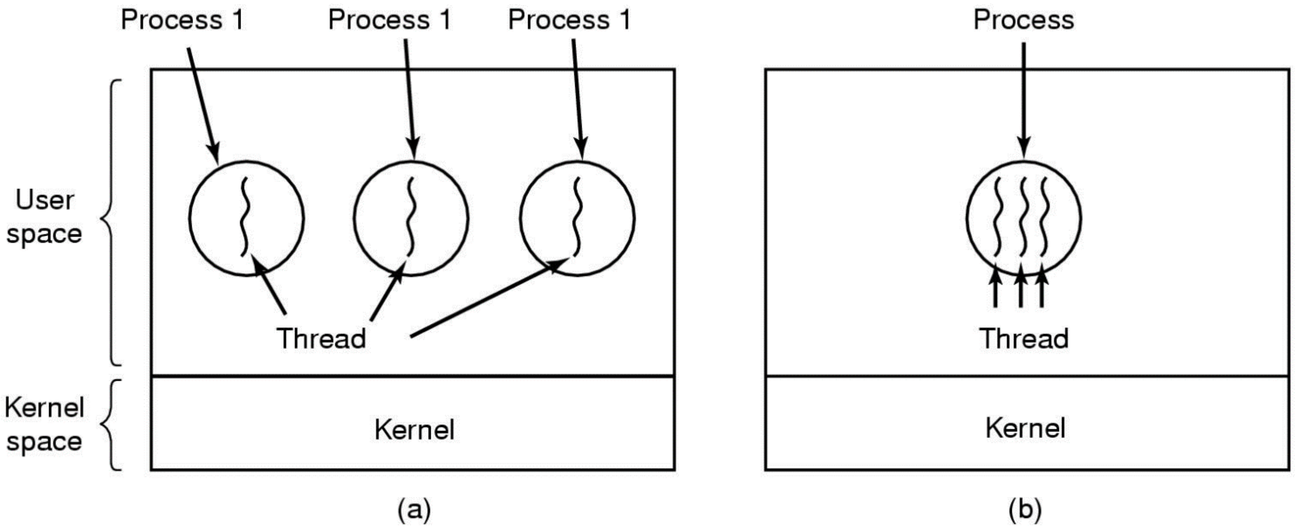
Per thread items
Program counter
Registers
Stack
State



single-threaded process

multithreaded process

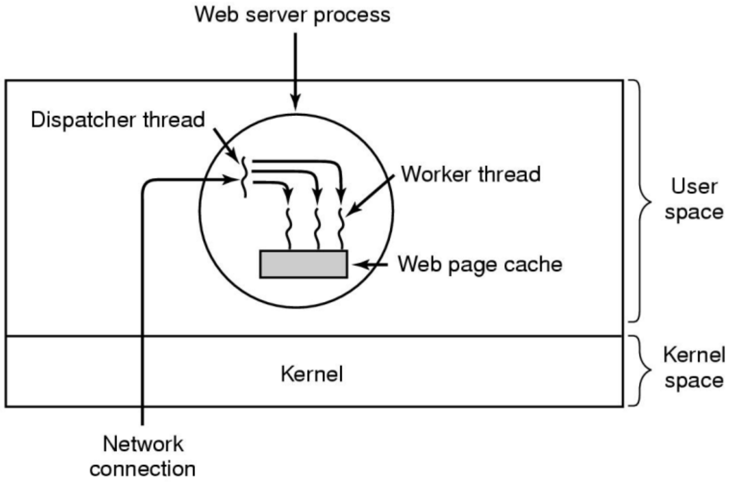
The Thread Model



Three processes each with one thread Vs One process with three threads

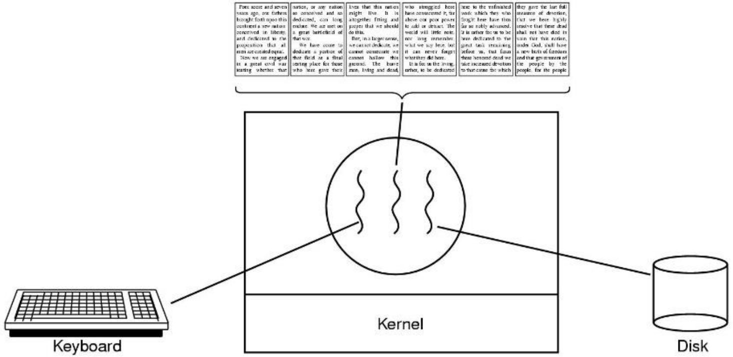
Multi-threaded programs

- Many software packages that run on modern OS are multi-threaded.
- A web browser might have
  - One thread display images or text
  - Another thread retrieves data from the network



Multi-threaded programs

- Many software packages that run on modern OS are multi-threaded.
- A word processor may have
  - A thread for displaying graphics
  - Another thread for responding to keystrokes form the user
  - A third thread for performing spelling and grammar checking



Multi-threaded programs

- Types of Web Server
    - Single-threaded web server: a client might have to wait for its request to be serviced.
    - Multi-threaded web server: less overhead in thread creation, concurrent service to multiple client.
- Many OS kernels are now multi-threaded
    - Several threads operates in the kernel
    - Each thread performs a specific task, such as managing devices or interrupt handling.

Benefits of multi-threaded programming

- Responsiveness
  - Multithreading an interactive application may allow a program to continue running even if part of it is blocked or doing a lengthy operation.
- Resource Sharing
  - Threads share the memory and the resources of the process to which they belong.
- Economy
  - Because threads in a process shares the resources, it is more economical to create and context-switch threads.
- Utilization of Multi-Processor Architectures
  - Threads may be running in parallel on different processors.

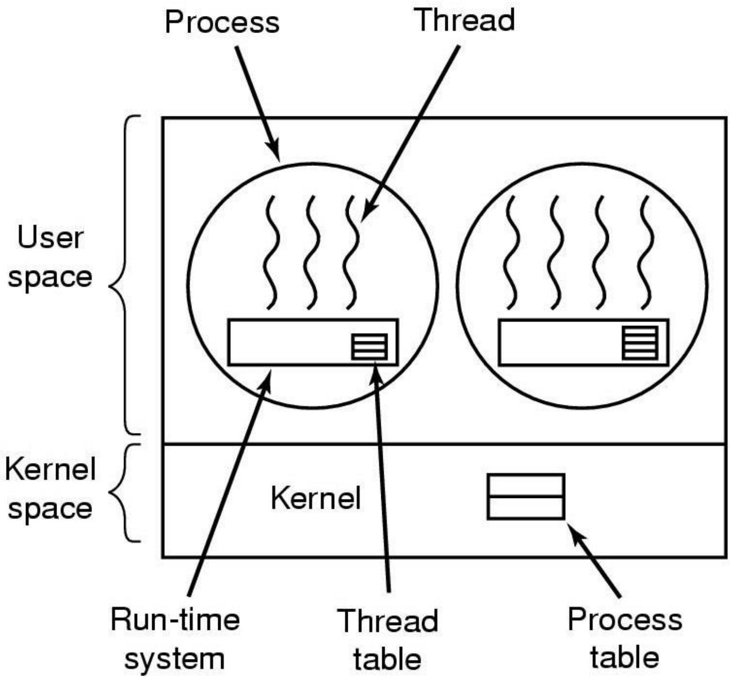
Two types of threads

- **User Thread**
  - User-level thread are threads that are visible to the programmer and are unknown to the kernel.
  - User thread are supported above the kernel and are managed without kernel support.
- Thread management done by user-level threads library
- Three primary thread libraries:
  - POSIX Pthreads
  - Win32 threads
  - Java threads

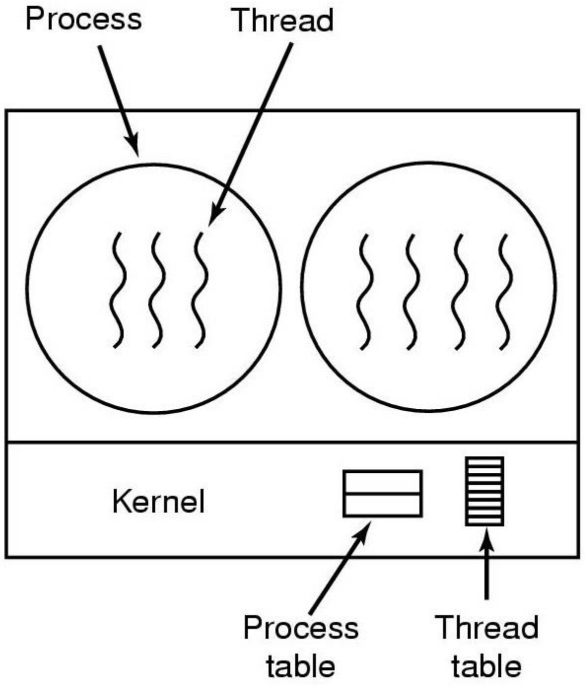
Two types of threads

- **Kernel Thread**
  - OS kernel supports and manages kernel-level threads
  - The threads are supported and managed directly by the operating system.
- Examples
  - Windows 10
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Implementing Threads in User Space



A user-level threads package



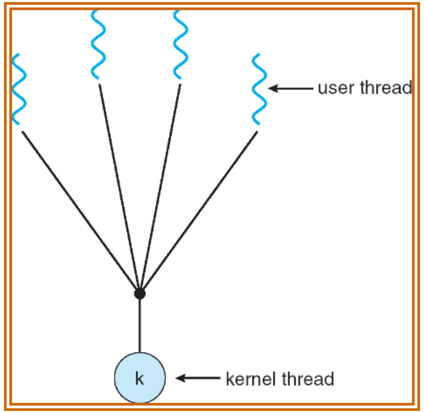
A threads package managed by the kernel

Multithreading Models

- A Relationship between user threads and kernel threads.
  - Many-to-One
  - One-to-One
  - Many-to-Many
  - Two Level Model

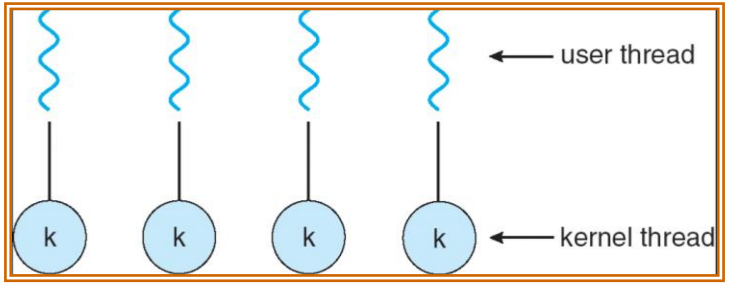
Many-to-One

- Many user-level threads mapped to single kernel thread
  - Thread management is done by the thread library in user space
  - Can create as many user threads as you wish.
  - The entire process will block when a thread makes a blocking system call.
  - Even on multiprocessors, threads are unable to run in parallel
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



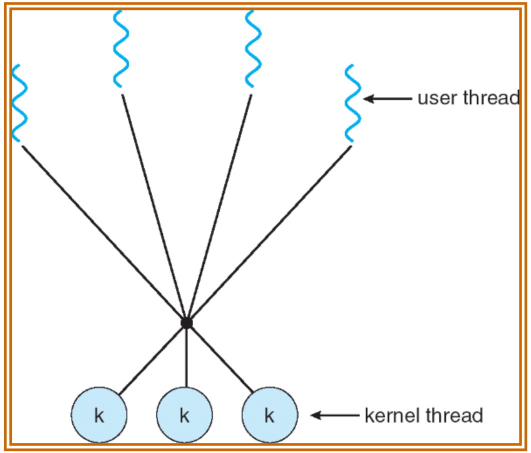
One-to-One

- Each user-level thread maps to a kernel thread
  - Provides more concurrency than the many-to-one model
  - Allows another thread to run when a thread is in blocking system call
  - Creating a user thread requires creating the corresponding kernel thread. (overhead)
  - The number of threads a process can create is smaller than many-to-one model. (careful not to create too many thread)
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later



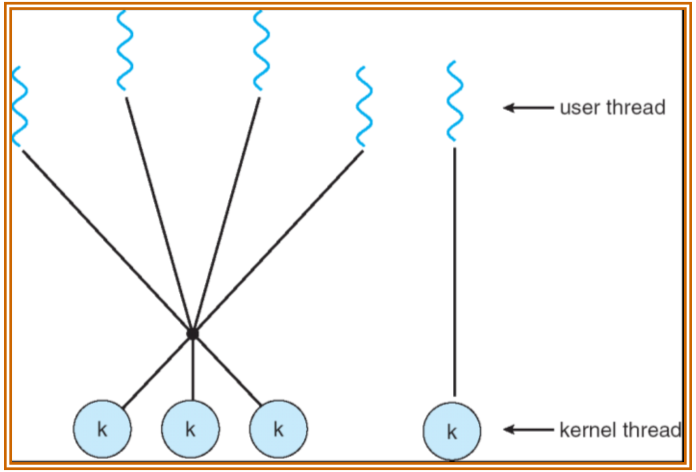
Many-to-Many Model

- Allows many user level threads to be mapped to smaller or equal kernel threads
  - Allows the OS to create a sufficient number of kernel threads
  - The number of kernel threads may be specific to either a application or machine
- Examples
  - Solaris prior to version 9
  - Windows NT/2000 with the ThreadFiber package



Two-Level Model

- One popular variation on many-to-many model
  - Similar to Many-to-Many model,
  - Many user-level threads are multiplexed to a smaller or equal number of kernel threads
  - But it allows a user thread to be **bound** to a kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



Thread Libraries

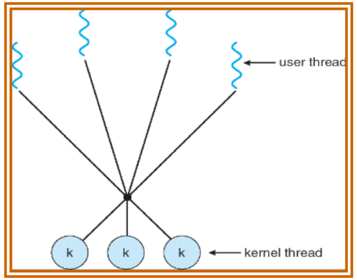
- A **thread library** provides the programmer an API for creating and managing threads
- Two primary ways to implement
  - to provide a library entirely in user space with no kernel support.
    - All code and data structures for the library exist in user space
    - Every function call executes in user mode, not in kernel mode
  - to implement a kernel-level library supported by OS
    - All code and data structures exist in kernel space
    - Invoking functions result in a system call to the kernel

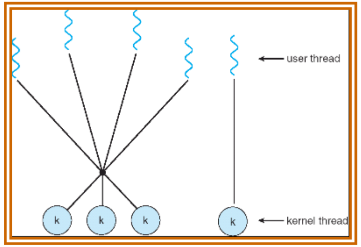
Thread Libraries

- A **thread library** provides the programmer an API for creating and managing threads
- **Three** main thread libraries
  - **POSIX Pthreads** - Solaris, Linux, Mac OS, Tru64 UNIX
  - **Win32 Thread** - Windows
  - **Java Thread** - Java

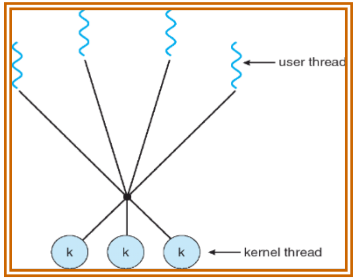
Light Weight Processes (LWP)

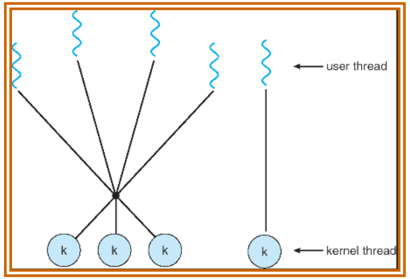
- Most popular mapping model - many-to-many or two-level mode
- Light Weight Process (LWP)
  - An intermediate data structure between user and kernel threads
  - A user thread is attached to a LWP
  - Each LWP is attached to a kernel thread
  - OS schedules the kernel threads (not processes) to run on CPU
  - If a kernel thread blocks, LWP blocks, and the user thread blocks





Light Weight Processes (LWP)





- To the user thread library, LWP appears to be a virtual CPU on which the application can schedule a user thread to run.
- The user thread library is responsible for scheduling among user threads to the LWPs. It is similar to the CPU scheduling in kernel.
- In general, context switching between user threads involves taking a user thread of its LWP and replacing it with another thread.

Threading Issues

- Semantics of fork() and exec() system calls
- Thread cancellation
- Signal handling
- Thread pools
- Scheduler activations

Semantics of fork() and exec()

- **fork()** - system call is used to create a duplicate process.
- **exec()** - system call is used to create a new separate process.
- Fork() starts a new process which is a copy of the one that calls it, while exec () replaces the current process image with another new one.
- Both parent and child processes are executed simultaneously in case of fork()
- In exec() control never returns to the original program unless there is an exec() error.

Thread Cancellation

- Terminating a thread before it has finished by other threads
  - Ex, multiple threads search DB, one thread returns result. The remaining thread might be canceled.
- Two general approaches to cancel the target thread
  - **Asynchronous cancellation** terminates the target thread immediately
  - **Deferred cancellation** allows the target thread to periodically check if it should be cancelled

Thread Cancellation

- **Asynchronous cancellation**
  - The difficulty with cancellation occurs in situation where
    - resources have been allocated to a canceled thread, or
    - where a thread is canceled while in the midst of updating data it is sharing with other threads
- **Deferred cancellation**
  - One thread indicates that target thread is to be canceled.
  - Cancellation occurs only after the target thread has checked a flag to determine if it should be canceled or not

Signal Handling

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred
- **Two types of signals**
  - Synchronous signals [illegal memory access, division by 0]
  - Asynchronous signals [Specific keystrokes (Ctrl-C), timer expire]
- What happen when a signal generated?
  - Signal is generated by particular event
  - Signal is delivered from kernel to a process
  - Signal is handled

Signal Handling

- Every signal may be handled by one of two possible handlers.
  - A default signal handler
  - A user-defined signal handler
- Every signal has a default signal handler that is run by the kernel
- The default action can be overridden by a user-defined signal handler
- Options when a signal occurs on a multi-threaded process
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

Thread Pools

- **Pool** of threads where they await work
- A process creates few threads at start up and place into a pool
- When receiving a request, server awakens a thread from the pool and passes the request to service
- Ones the thread completes its service, it returns to the pool and await more work.
- If the pool contains no available thread, the server waits until one becomes free.
- Thread pools are faster to service a request with an existing thread than create a new thread
- Allows the number of threads in the application(s) to be bound to the size of the pool

Scheduler Activations

- Both Many-to-Many and Two-level models require communication to maintain the appropriate number of kernel threads allocated to the application
- Scheduler activations provide upcalls - a communication mechanism from the kernel to the thread library
- This communication allows an application to maintain the correct number kernel threads

Summary

- A thread is a flow of control within process.
- Multithreaded process contains several different flows of control within the same address space.
- Benefits of multi-threading includes
  - Increased responsiveness, Resource sharing within the process
  - Economy, Ability to take advantage of multiprocessor architecture
- User-level thread are thread that are visible to the programmer and are unknown to the kernel.

Summary

- OS kernel supports and manages kernel-level threads
- Three types of models relates user and kernel threads
  - One-to-one, many-to-one, many-to-many
- Thread libraries provide the application programmer with an API for creating and managing threads
  - POSIX Pthreads, Win32 threads, Java threads
- Multithreaded programs introduces several issues
  - fork()/exec(), thread cancellation, signal handling, thread pools and schedule activation.

johnjose@iitg.ac.in

http://www.iitg.ac.in/johnjose/