Operating System

Dr. GuoJun LIU

Harbin Institute of Technology

http://guojunos.hit.edu.cn

Outline

- **Processes and Threads**
  - Multithreading
  - Thread Functionality
- **Types of Threads**
  - User-Level and Kernel-Level Threads
  - Other Arrangements

Chapter 04

Threads

线程

The basic idea is that the several components in any complex system will perform particular subfunctions that contribute to the overall function.

-- THE SCIENCES OF THE ARTIFICIAL, Herbert Simon

Turing Award 1975
Nobel Prize in Economics 1978
National Medal of Science 1986
von Neumann Theory Prize 1988

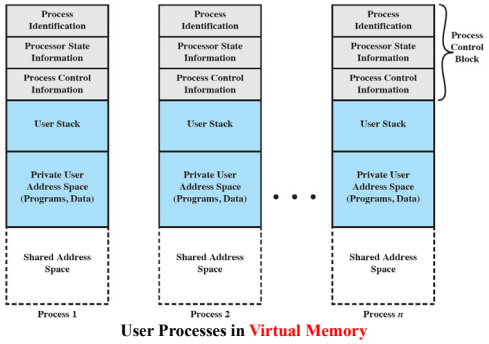
Learning Objectives

- **Understand the distinction between process and thread**
- **Describe the basic design issues for threads**
- **Explain the difference between user-level threads and kernel-level threads**

Processes and Threads

- **Resource Ownership**
  - a virtual address space to hold the process image
  - may be allocated control or ownership of resources
    - main memory
    - I/O channels and devices
    - files
  - OS performs a protection function to prevent unwanted interference between processes with respect to resources
- **Scheduling/Execution**
  - Follows an execution path that may be interleaved with other processes
  - an execution state
    - Running, Ready, etc.
  - a dispatching priority
  - is the entity that is scheduled and dispatched by the OS

Structure of Process Images

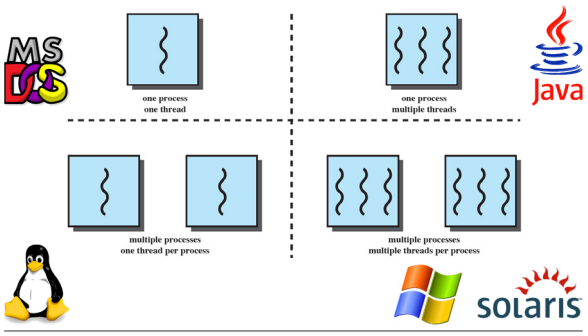


User Processes in Virtual Memory

Processes and Threads

- **Thread** or **lightweight process LWT**
  - The unit of **dispatching** is referred to as a thread or lightweight process
- **Process** or **Task**
  - The unit of **resource ownership** is referred to as a process or task
- **Multithreading**
  - The ability of an OS to support multiple, concurrent paths of execution within a single process

Threads and Processes



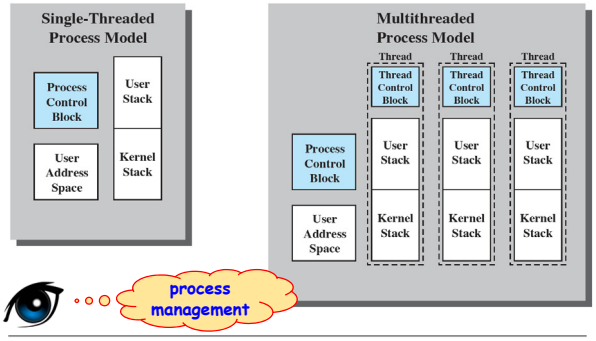
Processes

- **The unit of resource allocation**
  - **A virtual address space** that holds the **process image**
- **The unit of protection**
  - processors
  - other processes
    - for interprocess communication
  - files
  - I/O resources
    - devices and channels

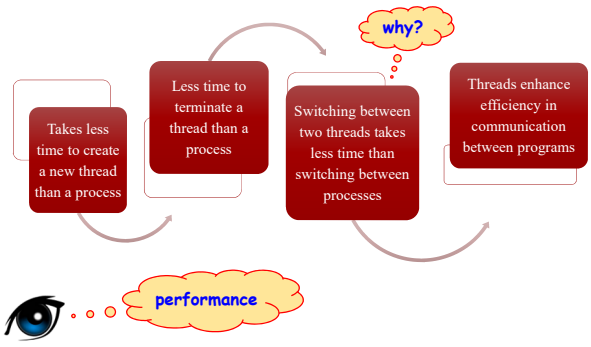
One or More Threads in a Process

- **Each thread has**
  - an execution state
    - Running, Ready, etc.
  - saved **thread context** when not running
    - one way to view a thread is **as an independent program counter** operating within a process
  - 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

Threads vs. Processes



Key Benefits of Threads



Thread Use in a Single-User System

- **Foreground and background work**
  - a spreadsheet program
- **Asynchronous processing**
  - as a protection against power failure
- **Speed of execution**
  - A multithreaded process can compute one batch of data while reading the next batch from a device
- **Modular program structure**
  - Programs that involve a variety of activities or a variety of sources and destinations of input and output

Thread Functionality

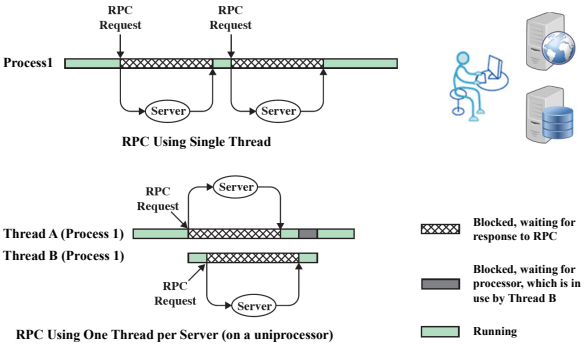
- **Scheduling and dispatching is done on a thread basis**
- **Most of the state information dealing with execution is maintained in thread-level data structures**
- **Several actions that affect all of the threads in a process**
  - the OS must manage at the **process level**
    - **suspending** a process involves suspending all threads of the process
    - **termination** of a process terminates all threads within the process

Thread Execution States

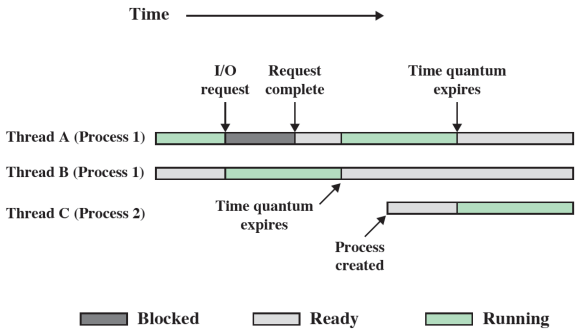
- **Key states for a thread**
  - Running
  - Ready
  - Blocked
- **Thread operations associated with a change in thread state**
  - **Spawn**
  - Block
  - Unblock
  - Finish

- **Spawn**
  - when a new process is spawned, a thread for that process is also spawned
  - a thread within a process may spawn another thread within the same process
  - The new thread is provided with its own register context and stack space and placed on the ready queue

Performance Benefits of Threads

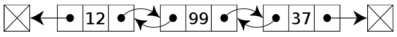


Multithreading on a Uniprocessor

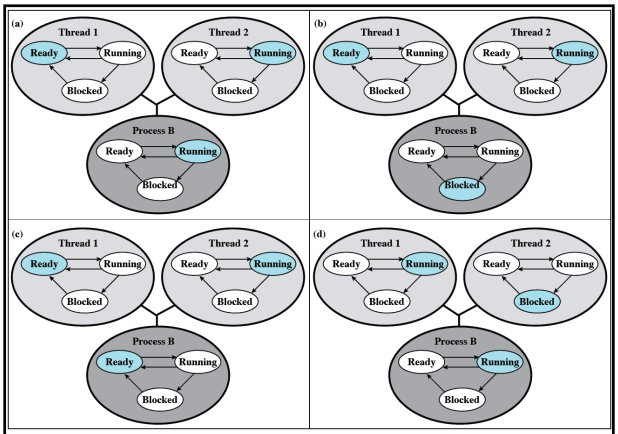


Thread Synchronization

- **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

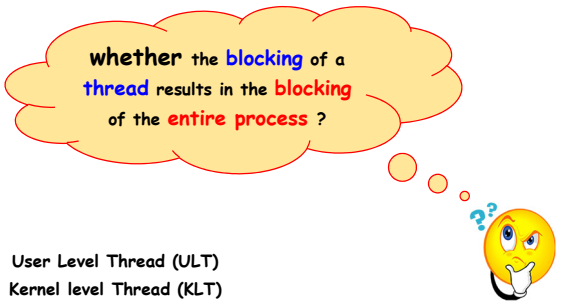


if two threads each try to add an element to a doubly linked list at the same time
- one element may be **lost**
- the list may end up **malformed**



Examples of the Relationships Between User-Level Thread States and Process States

Types of Threads



User Level Thread (ULT)

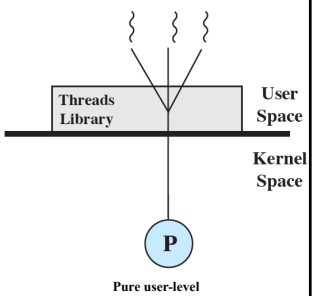
Kernel level Thread (KLT)

Advantages of ULTs

- **Thread switching does not require kernel mode privileges**
  - This saves the overhead of two mode switches
    - user to kernel; kernel back to user
- **Scheduling can be application specific**
  - The scheduling algorithm can **be tailored to the application** without disturbing the underlying OS scheduler
- **ULTs can run on any OS**
  - The threads library is a set of application-level functions shared by all applications

User-Level Threads (ULTs)

- **All thread management is done by the application**
  - in user space
  - within a single process
- **The kernel is not aware of the existence of threads**

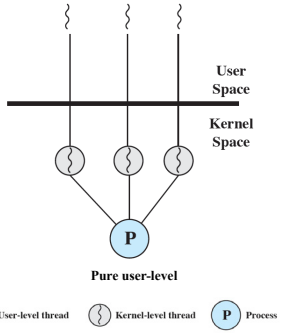


Disadvantages of ULTs

- **Disadvantages**
  - when a ULT executes a **system call**, not only is that thread blocked, but also all of the threads within the process are blocked
  - In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing
- **Overcoming**
  - Writing an application as **multiple processes** rather than multiple threads
    - Each switch becomes a process switch rather than a thread switch, resulting in much greater overhead
  - **Jacketing**
    - converts a **blocking** system call into a **non-blocking** system call

Kernel-Level Threads (KLTs)

- **Thread management is done by the kernel**
  - no thread management is done by the application
  - Windows is an example of this approach



Advantages of KLTs

- **The kernel can simultaneously schedule multiple threads from the same process on multiple processors**
- **If one thread in a process is blocked, the kernel can schedule another thread of the same process**
- **Kernel routines can be multithreaded**

Disadvantage of KLTs

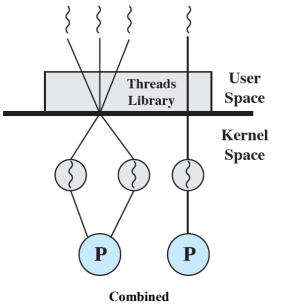
- **The transfer of control from one thread to another within the same process requires a mode switch to the kernel**

Table Thread and Process Operation Latencies (μs)

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Combined Approaches

- **Thread creation is done in the user space**
- **Bulk of scheduling and synchronization of threads is by the application**
- **The multiple ULTs are mapped onto some number of KLTs**
- **The programmer may adjust the number of KLTs**



Relationship Between Threads and Processes

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Summary

- **Process**
  - resource ownership
- **Thread**
  - program execution
- **User-level threads**
  - created and managed by a threads library that runs in the user space of a process
  - a mode switch is not required to switch from one thread to another
  - only a single user-level thread within a process can execute at a time
  - if one thread blocks, the entire process is blocked
- **Kernel-level threads**
  - threads within a process that are maintained by the kernel
  - a mode switch is required to switch from one thread to another
  - multiple threads within the same process can execute in parallel on a multiprocessor
  - blocking of a thread does not block the entire process