目录 Contents

Threads and Kernel Architecture

3.1 Process and Threads

Two characteristics of processes

- Resource ownership - process includes a virtual address space to hold the process image.
  - the 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
  - a process has an execution state (Running, Ready, etc.) and a dispatching priority, and is scheduled and dispatched by the OS

> These two characteristics are treated independently by the operating system.

3.1 Process and Threads

1 Multithreading

- The unit of dispatching is referred to as a **thread** or **lightweight process**.
- 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.

Single-Threaded Approaches

- A single thread of execution per process, in which the concept of a thread is not recognized, is referred to as a single-threaded approach
  - MS-DOS is an example.

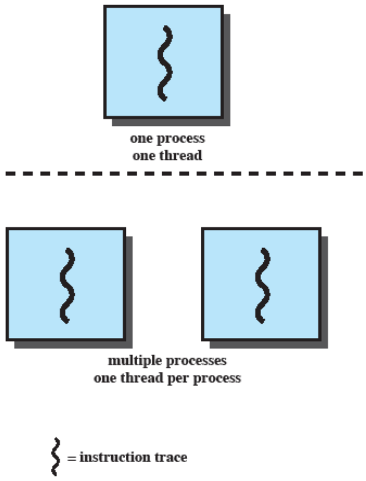


Figure 4.1 Threads and Processes [ANDE97]

Multithreaded Approaches

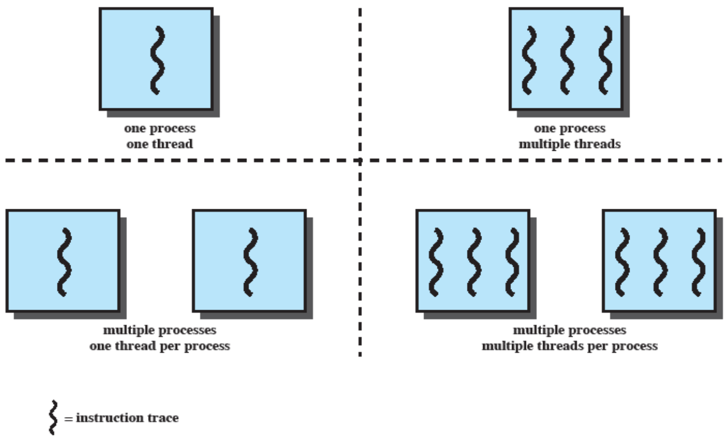


Figure 4.1 Threads and Processes [ANDE97]

Process in Multithreading

- In a multithreaded environment, a process is defined as the unit of **resource allocation** and a unit of **protection**:
  - resource allocation: Have a virtual address space which holds the process image
  - Protected access to:
    - processors
    - other processes (for interprocess communication)
    - files
    - I/O resources (devices and channels)

Thread in Multithreading

- Within a process, there may be one or more threads, each with the following:
  - an execution state (Running, Ready, etc.)
  - saved thread context when not running
  - 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

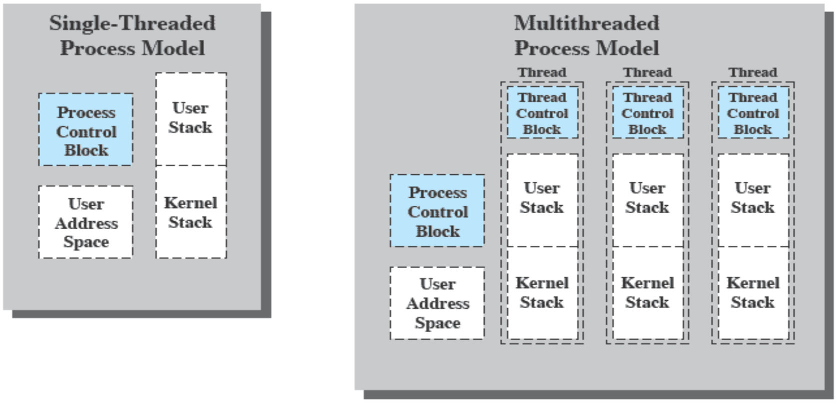


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefits of Threads

- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Less time to switch between two threads within the same process
- Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel

Multithreaded Scenarios

- Foreground and background work: eg. one thread display and read user input, while another executes user commands
- Asynchronous processing: eg. periodic backup
- Speed of execution: eg. one thread may be blocked, another thread may be executing
- Modular program structure: eg. Programs that involve a variety of activites or a variety of sources and destination of input and output may be earier to design and implement using threads

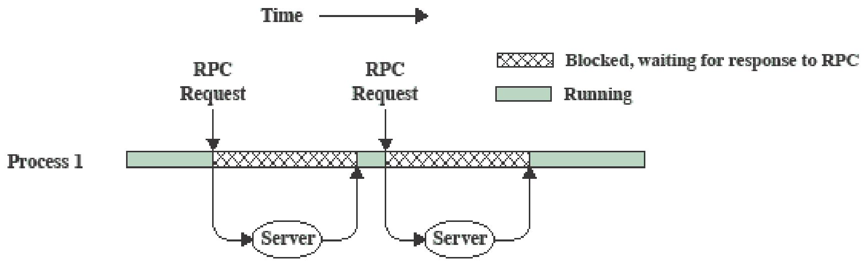
3.1 Process and Threads

2 Thread States

- The key states for a thread are:
  - Running
  - Ready
  - Blocked
- Thread operations associated with a change in thread state are:
  - Spawn: Spawn another thread
  - Block
  - Unblock
  - Finish: Deallocate register context and stacks

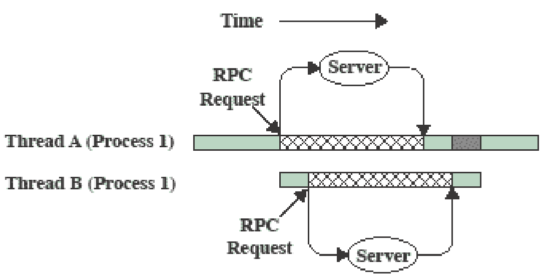
Example—One Thread in One Process

- A program that performs two remote procedure calls (RPCs) to two different hosts to obtain a combined result
  - RPCs use single thread
  - RPCs use multiple threads



(a) RPC Using Single Thread

Example—Multiple Threads in One Process



(b) RPC Using One Thread per Server (on a uniprocessor)

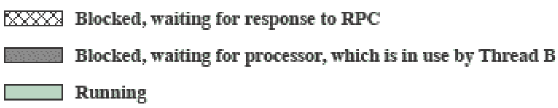


Figure 4.3 Remote Procedure Call (RPC) Using Threads

Multiple Threads Within Multiple Processes

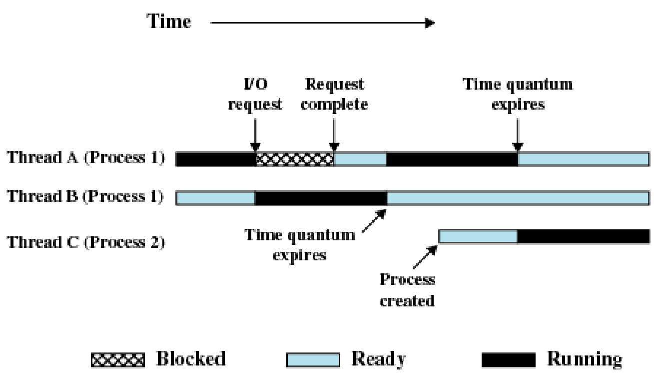


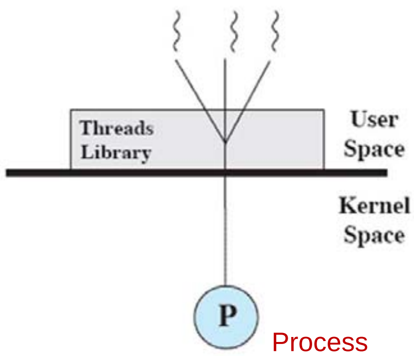
Figure 4.4 Multithreading Example on a Uniprocessor

3.2 Thread Implementation

1 User Level Threads (ULTs)

- All thread management is done by the application
- The kernel is not aware of the existence of threads

> The threads library contains code for creating and destroying threads, for passing messages and data between threads, for scheduling thread execution, and for saving and restoring thread contexts.



(a) Pure user-level

ULT States and Process States

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Advantages and Disadvantages of ULTs

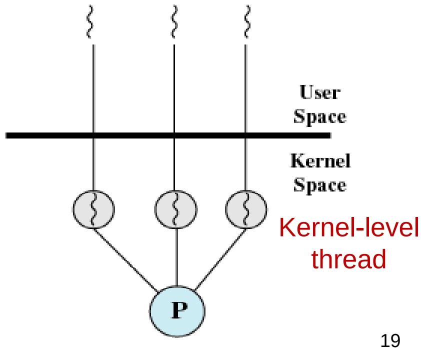
- Advantages of ULTs
  - Thread switching does not require kernel mode privileges
  - Scheduling can be application specific
  - ULTs can run on any OS
- Disadvantages of ULTs
  - In a typical OS, many system calls are blocking, as a result, when a ULT executes a system call, not only is that thread blocked, but all of the threads within the process are blocked
  - In a pure ULT strategy, a multithreaded application cannot take advantage of multiprocessing. A kernel assigns one process to only one processor at a time. Therefore, only a single thread within a process can execute at a time.

3.2 Thread Implementation

2 Kernel-Level Threads (KLTs)

- Thread management is done by the kernel，no thread management is done by the application
- Scheduling is done on a thread basis

> - Kernel maintains context information for the process and the threads
> - Windows is an example of this approach



(b) Pure kernel-level

Advantages and Disadvantages of KLTs

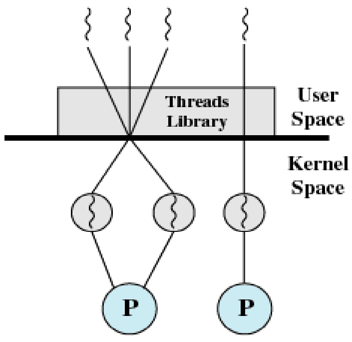
- Advantages of KLTs
  - The kernel can simultaneously schedule multiple threads from the same process on multiple processors
  - If one thread in a process is blocked, the kernel can schedule another thread of the same process
  - Kernel routines can be multithreaded
- Disadvantages of KLTs
  - The transfer of control from one thread to another within the same process requires a mode switch to the kernel.

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

3.2 Thread Implementation

3 Combined Approaches

- Thread creation done in the user space, as is the bulk of scheduling and synchronization of threads within application
- The multiple ULTs from a single application are mapped onto some (smaller or equal) number of KLTs.
- The programmer may adjust the number of KLTs
  - Example is Solaris



(c) Combined

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

Benefited Applications

- Multithreaded native applications
  - characterized by having a small number of highly threaded processes
- Multiprocess applications
  - characterized by the presence of many single-threaded processes
- Java applications
- Multiinstance applications
  - multiple instances of the application in parallel

3.3 Process and Thread In Typical OS

1 Windows Process and Thread Management

- Processes and services provided by the Windows Kernel are relatively simple and general purpose.

- Processes are implemented as objects
- A process can be created as a new process or a copy of an existing process
- An executable process may contain one or more threads
- Both processes and thread objects have built-in synchronization capabilities

Relationship Between Process and Resource

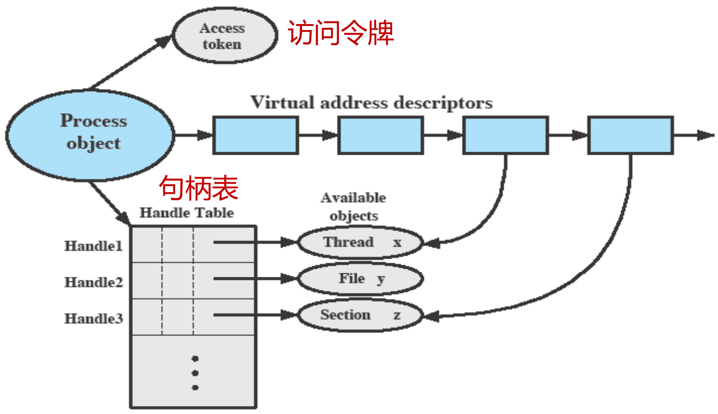
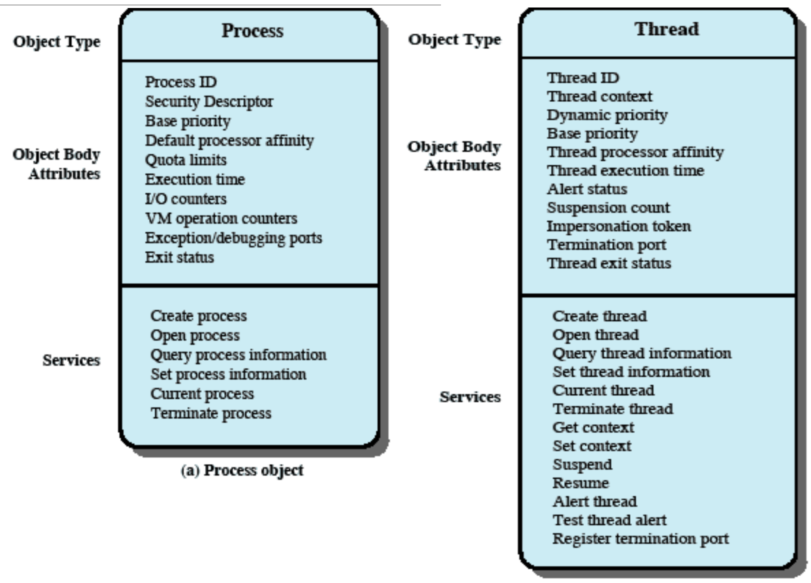


Figure 4.12 A Windows Process and Its Resources

Process and Thread Objects

- Windows makes use of two types of process-related objects:
  - Processes: an entity corresponding to a user job or application that owns resources
  - Threads: a dispatchable unit of work that executes sequentially and is interruptible

Windows Process and Thread Objects



(a) Process object

(b) Thread object

Multithreaded Process

- Achieves concurrency without the overhead of using multiple processes
- Threads within the same process can exchange information through their common address space and have access to the shared resources of the process
- Threads in different processes can exchange information through shared memory that has been set up between the two processes

Thread States

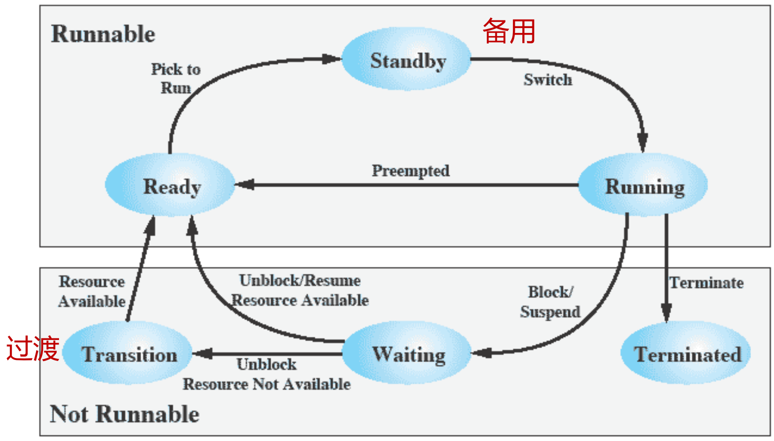


Figure 4.14 Windows Thread States

3.3 Process and Thread In Typical OS

2 Solaris Process and Thread Management

- Solaris makes use of four separate thread-related concepts.

- **Process**: includes the user's address space, stack, and process control block
- **User-level Threads**: a user-created unit of execution within a process
- **Lightweight Processes (LWP)**: a mapping between ULTs and kernel threads
- **Kernel Threads**: fundamental entities that can be scheduled and dispatched to run on one of the system processors

Processes and Threads in Solaris

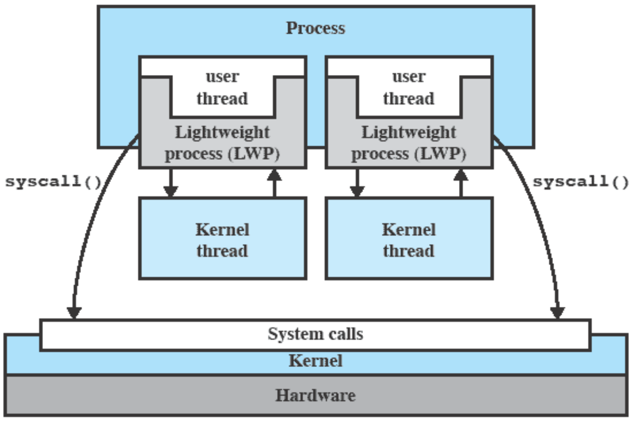


Figure 4.15 Processes and Threads in Solaris [MCDO07]

Traditional Unix vs Solaris

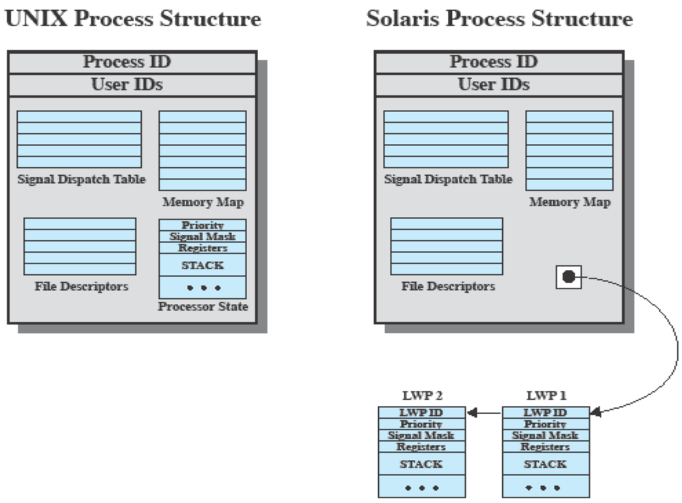


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

Solaris Thread States

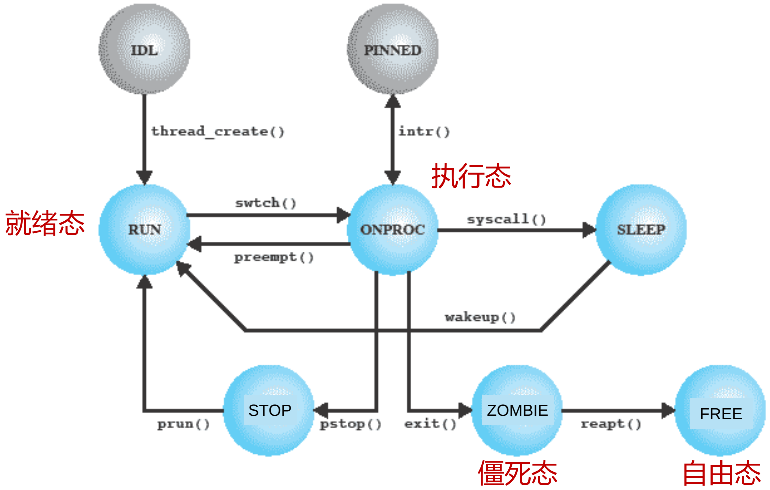


Figure 4.17 Solaris Thread States [MCDO07]

Terminology

- thread(lightweight process); process(task)
- multithreading
- user level thread (ULT);
- kernel level thread (KLT)