Threads, SMP, and Microkernels

Chapter 4

1

Process

- Resource ownership - process is allocated a virtual address space to hold the process image
- Scheduling/execution- follows an execution path that may be interleaved with other processes
- These two characteristics are treated independently by the operating system

2

Process

- Dispatching is referred to as a thread
- Resource ownership is referred to as a process or task

3

Multithreading

- Operating system supports multiple threads of execution within a single process
- MS-DOS supports a single thread
- UNIX supports multiple user processes but only supports one thread per process
- Windows 2000, Solaris, Linux, Mach, and OS/2 support multiple threads

4

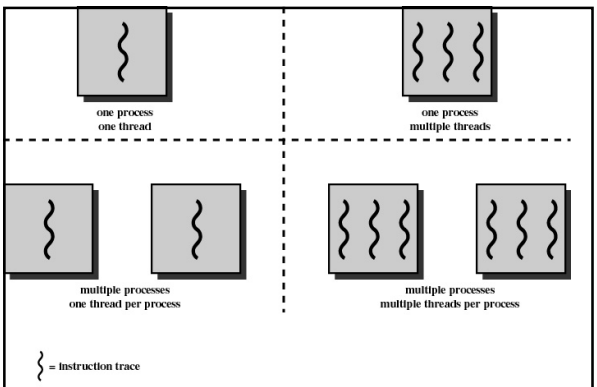


Figure 4.1 Threads and Processes [ANDE97]

Process

- Have a virtual address space which holds the process image
- Protected access to processors, other processes, files, and I/O resources

6

Thread

- An execution state (running, ready, etc.)
- Saved thread context when not running
- Has 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

7

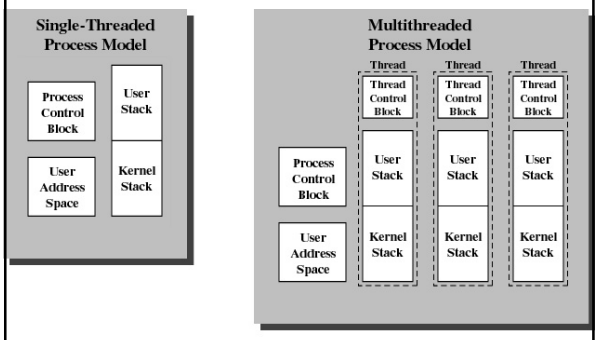


Figure 4.2 Single Threaded and Multithreaded Process Models

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

9

Uses of Threads in a Single-User Multiprocessing System

- Foreground to background work
- Asynchronous processing
- Speed execution
- Modular program structure

10

Threads

- Suspending a process involves suspending all threads of the process since all threads share the same address space
- Termination of a process, terminates all threads within the process

11

Thread States

- States associated with a change in thread state
  - Spawn
    - Spawn another thread
  - Block
  - Unblock
  - Finish
    - Deallocate register context and stacks

12

Remote Procedure Call Using Threads

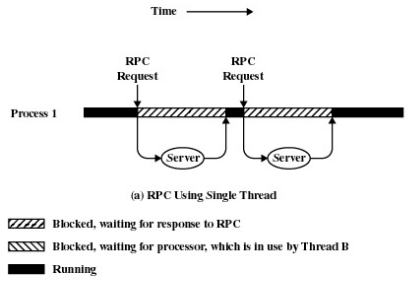


Figure 4.3 Remote Procedure Call (RPC) Using Threads

Remote Procedure Call Using Threads

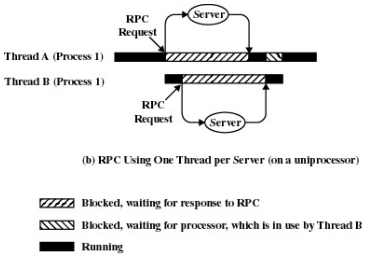


Figure 4.3 Remote Procedure Call (RPC) Using Threads

14

User-Level Threads

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

15

Kernel-Level Threads

- W2K, Linux, and OS/2 are examples of this approach
- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis

16

Combined Approaches

- Example is Solaris
- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads done in the user space

17

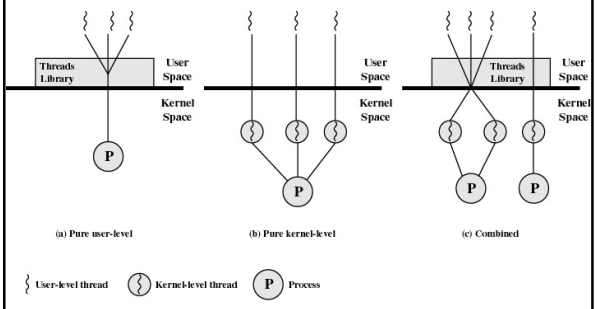


Figure 4.6 User-Level and Kernel-Level Threads

Relationship Between Threads and Processes

Threads:Process	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, OS/2, OS/390, MACH

19

Relationship Between Threads and Processes

Threads:Process	Description	Example Systems
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:M	Combines attributes of M:1 and 1:M cases	TRIX

20

Categories of Computer Systems

- Single Instruction Single Data (SISD)
  - single processor executes a single instruction stream to operate on data stored in a single memory
- Single Instruction Multiple Data (SIMD)
  - each instruction is executed on a different set of data by the different processors

21

Categories of Computer Systems

- Multiple Instruction Single Data (MISD)
  - a sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. Never implemented
- Multiple Instruction Multiple Data (MIMD)
  - a set of processors simultaneously execute different instruction sequences on different data sets

22

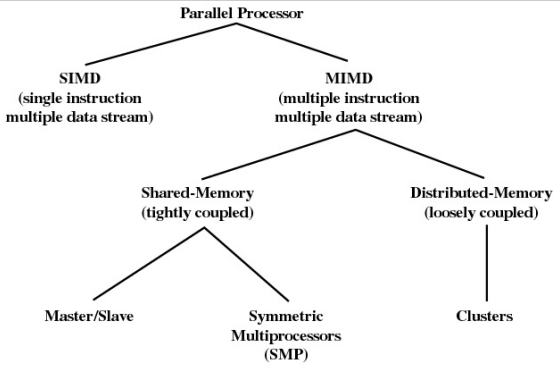


Figure 4.7 Parallel Processor Architectures

Symmetric Multiprocessing

- Kernel can execute on any processor
- Typically each processor does self-scheduling form the pool of available process or threads

24

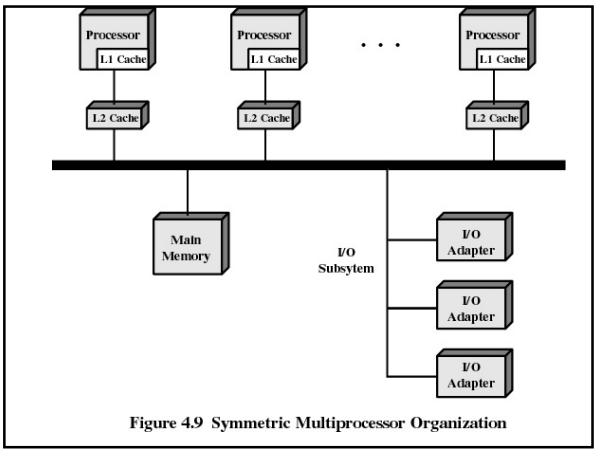


Figure 4.9 Symmetric Multiprocessor Organization

Multiprocessor Operating System Design Considerations

- Simultaneous concurrent processes or threads
- Scheduling
- Synchronization
- Memory Management
- Reliability and Fault Tolerance

26

Microkernels

- Small operating system core
- Contains only essential operating systems functions
- Many services traditionally included in the operating system are now external subsystems
  - device drivers
  - file systems
  - virtual memory manager
  - windowing system
  - security services

27

Benefits of a Microkernel Organization

- Uniform interface on request made by a process
  - All services are provided by means of message passing
- Extensibility
  - Allows the addition of new services
- Flexibility
  - New features added
  - Existing features can be subtracted

28

Benefits of a Microkernel Organization

- Portability
  - Changes needed to port the system to a new processor is changed in the microkernel - not in the other services
- Reliability
  - Modular design
  - Small microkernel can be rigorously tested

29

Benefits of Microkernel Organization

- Distributed system support
  - Messages are sent without knowing what the target machine is
- Object-oriented operating system
  - Components are objects with clearly defined interfaces that can be interconnected to form software

30

Microkernel Design

- Low-level memory management
  - mapping each virtual page to a physical page frame
- Inter-process communication
- I/O and interrupt management

31

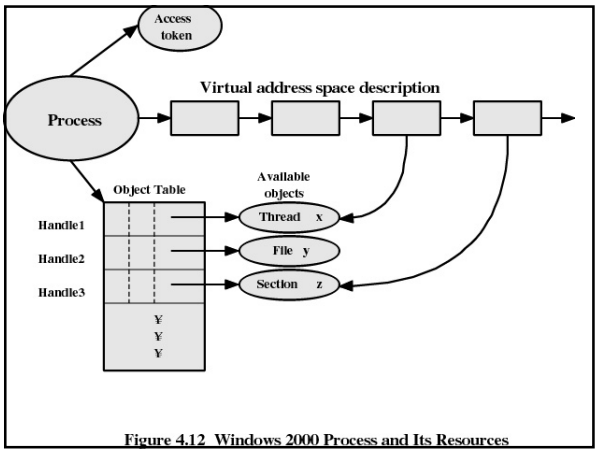
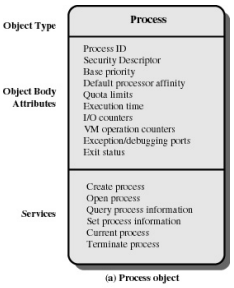


Figure 4.12 Windows 2000 Process and Its Resources

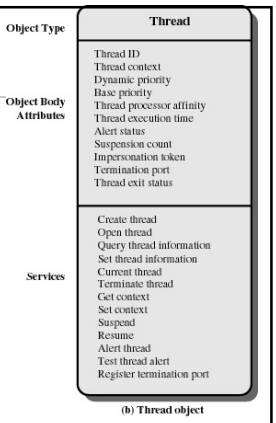
Windows 2000 Process Object



(a) Process object

33

Windows 2000 Thread Object



(b) Thread object

34

Windows 2000 Thread States

- Ready
- Standby
- Running
- Waiting
- Transition
- Terminated

35

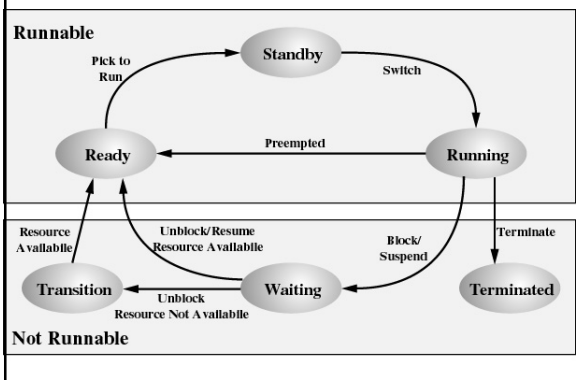


Figure 4.14 Windows 2000 Thread States

Solaris

- Process includes the user's address space, stack, and process control block
- User-level threads
- Lightweight processes
- Kernel threads

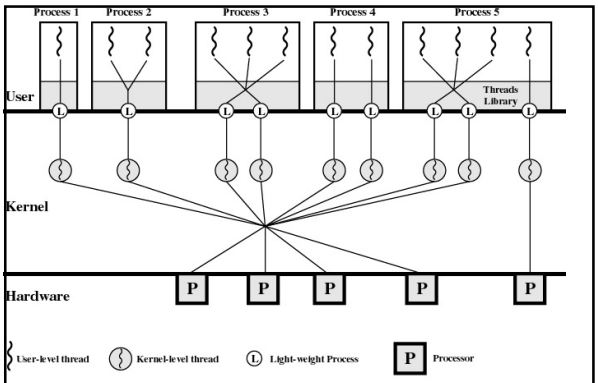


Figure 4.15 Solaris Multithreaded Architecture Example

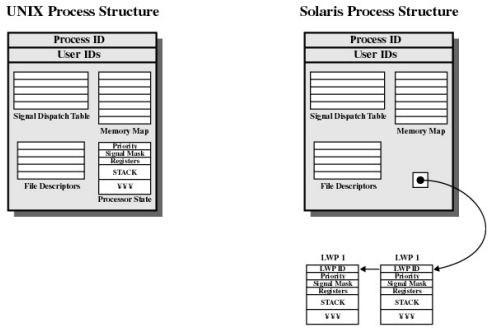


Figure 4.16 Process Structure in Traditional UNIX and Solaris [LEWI96]

Solaris Thread Execution

- Synchronization
- Suspension
- Preemption
- Yielding

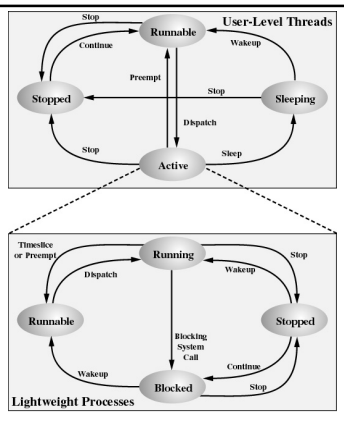


Figure 4.17 Solaris User-Level Thread and LWP States

Linux Process

- State
- Scheduling information
- Identifiers
- Interprocess communication
- Links
- Times and timers
- File system
- Virtual memory
- Processor-specific context

Linux States of a Process

- Running
- Interruptable
- Uninterruptable
- Stopped
- Zombie

43

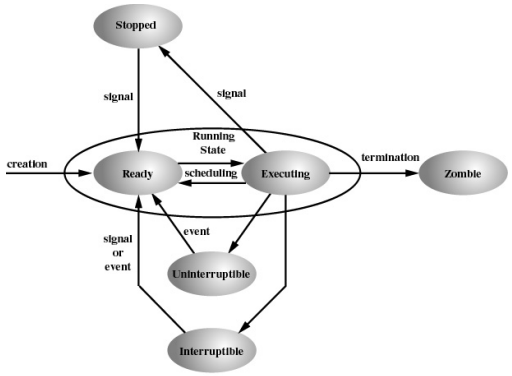


Figure 4.18 Linux Process/Thread Model