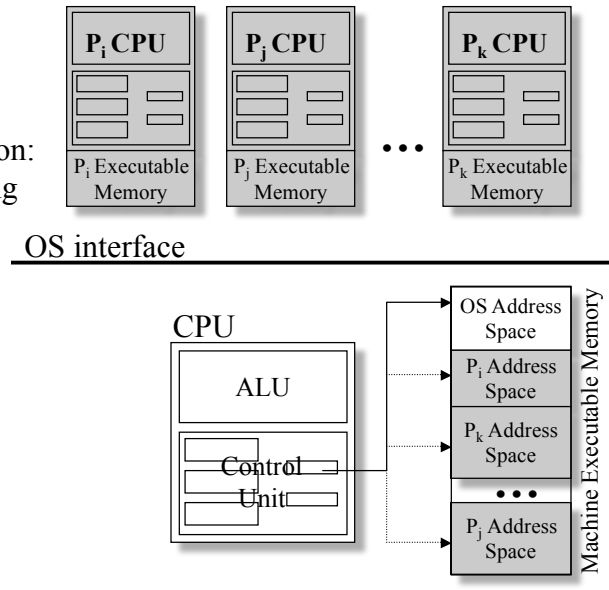
Implementing Processes, Threads, and Resources

Implementing the Process Abstraction

Ideal Abstraction: As if using an Actual machine



Modern process

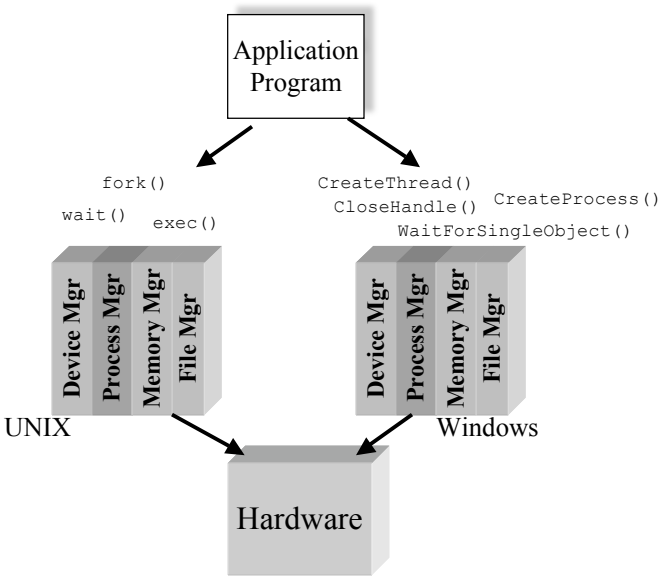
Slide 6-3

"The value of the modern process model is that it enables the programmer to design software so that various parts of the computation can work together as a set of threads within a single modern process framework."

Classic process can work together but they do not share a customized computational framework

External View of the Process Manager

Slide 6-4



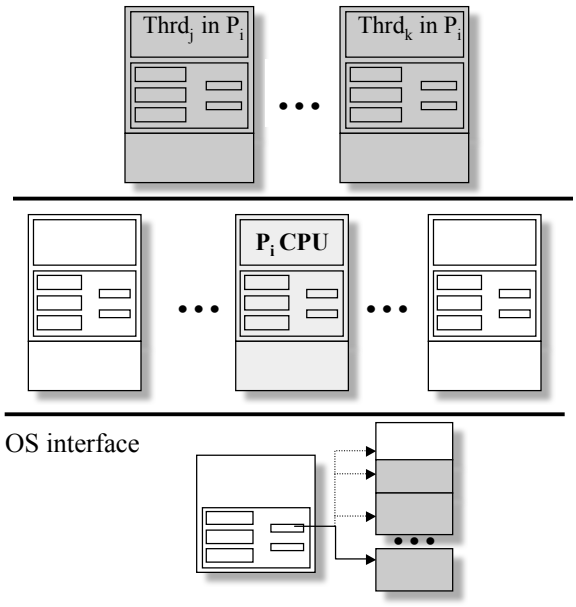
Process Manager Responsibilities

Slide 6-5

- Define & implement the essential characteristics of a process and thread
  - Algorithms to define the behavior
  - Data structures to preserve the state of the execution
- Define what "things" threads in the process can reference – the *address space* (most of the "things" are memory locations)
- Manage the resources used by the processes/threads
- Tools to create/destroy/manipulate processes & threads
- Tools to time-multiplex the CPU – Scheduling the (Chapter 7)
- Tools to allow threads to synchronization the operation with one another (Chapters 8-9)
- Mechanisms to handle deadlock (Chapter 10)
- Mechanisms to handle protection (Chapter 14)

Modern Processes and Threads

Slide 6-6



Modern Threads

User Space Threads - underlying Os implements classic processes and the user space thread library executes on top of the OS abstract machine to multiprogram the threads. (Mach C and POSIX threads)

Kernel Threads - OS time-multiplexes the execution of threads instead of processes. Therefore when one thread blocks the other threads can still execute. (Windows)

Resources

Any element of the abstract machine that a process can request and can cause the process to be blocked if not available.

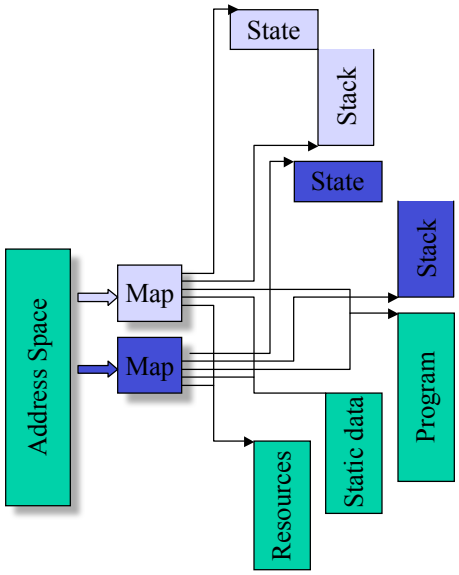
If device is allocated to a process then it is configured into the abstract machine for process

Multiple resource managers - hardware devices, processor, abstract synch resources, primary memory, and files

Each resource manager must present a common behavior described by a general model

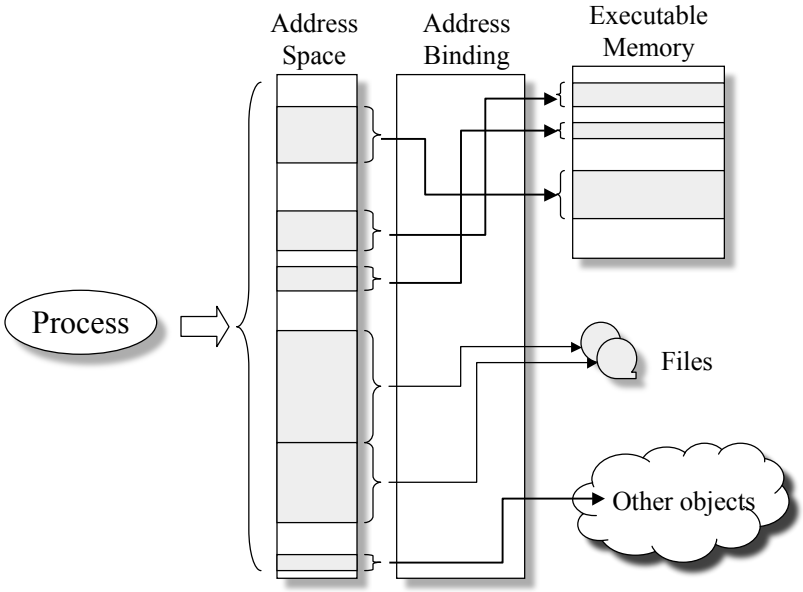
Processes &Threads

Slide 6-9



The Address Space

Slide 6-10



Building the Address Space

- Some parts are built into the environment
  - Files
  - System services
- Some parts are imported at runtime
  - Mailboxes
  - Network connections
- Memory addresses are created at compile (and run) time

The OS

- Abstract machine interface
  - Host hardware instruction set and set of functions exported by OS
- Unix and Windows most widely used
- Unix - POSIX interface (standard)

Modern Process Framework

Slide 6-13

- Thread-based computation is executed within this framework
- Modern Process structure
  - Address space
  - Program
  - Data - shared by threads
  - Resources
    - Threads share the resources that have been allocated to the process
  - Process Id

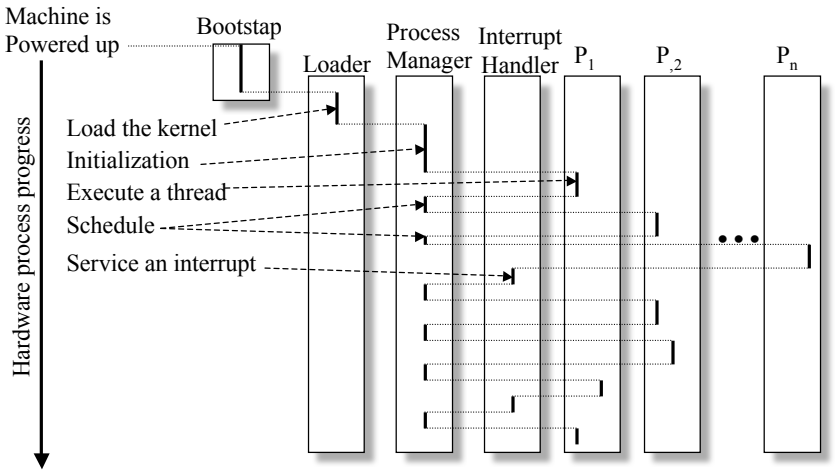
Threads

Slide 6-14

- Active Element
- Threads
  - Host process environment
  - Thread-specific data (at least a stack)
  - Thread ID

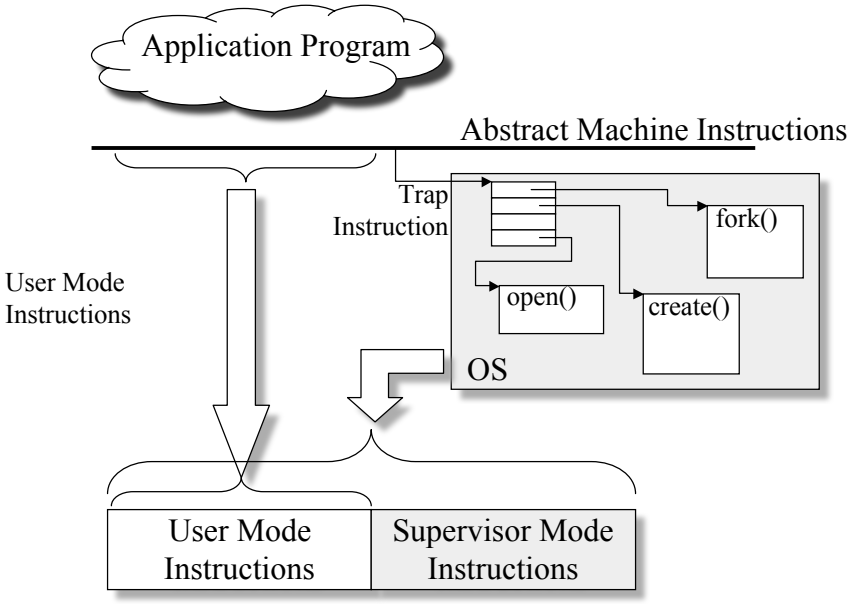
Slide 6-15

Tracing the Hardware Process



Slide 6-16

The Abstract Machine Interface



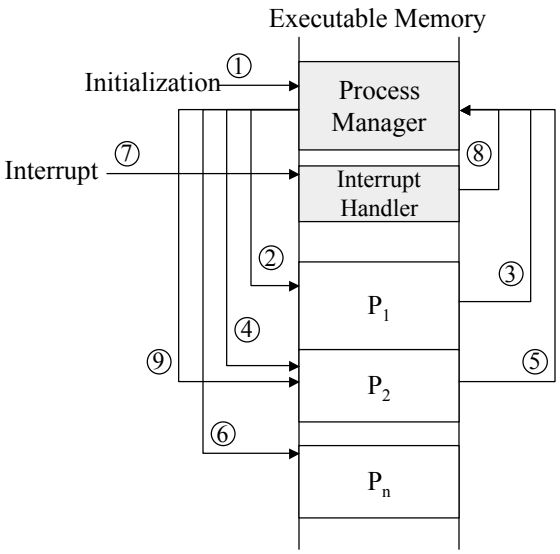
Abstract Machine Instruction Set

ALU - load, store, add…
Control Unit - branch, procedure_call…
Trap - create_process(), open_file()…

Linux 2.4x
- exports over 200 functions
- 2.5 million lines of code

Windows NT/2000/XP
- exports over 2,000 functions
- over 25 million lines of code

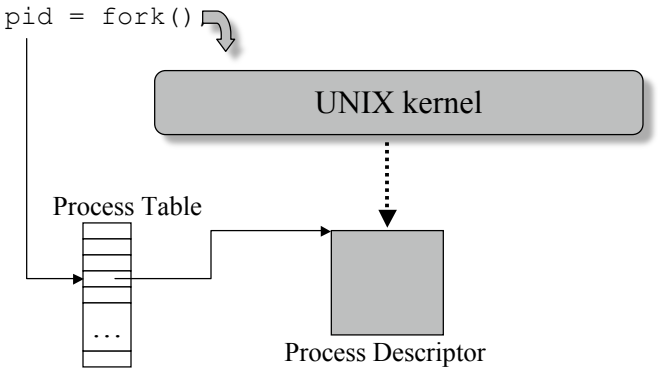
Context Switching



Process Descriptors

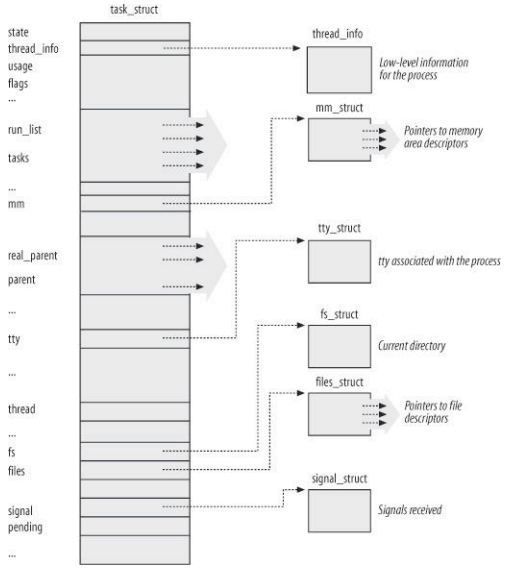
- OS creates/manages process abstraction
- Descriptor is data structure for each process
  - Register values
  - Logical state
  - Type & location of resources it holds
  - List of resources it needs
  - Security keys
  - Process ID, parent process

Creating a Process in UNIX



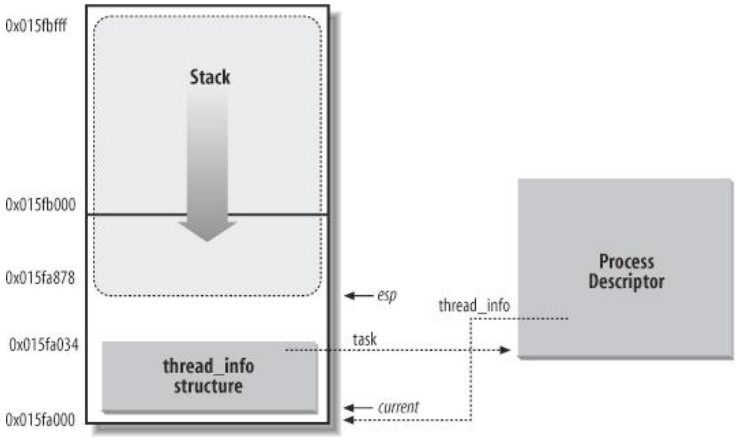
Linux Process Descriptor

Slide 6-21



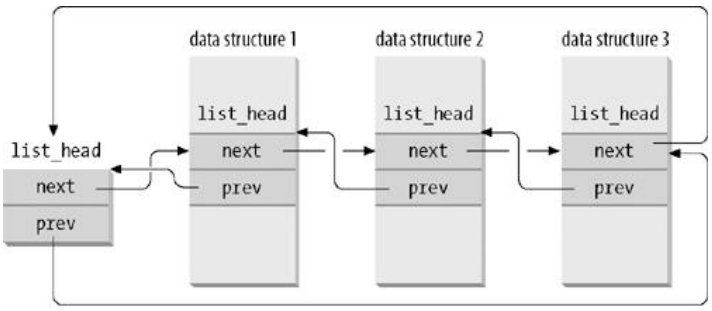
Linux Process Descriptor

Slide 6-22



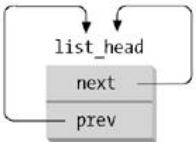
Linux Process Descriptor

Slide 6-23



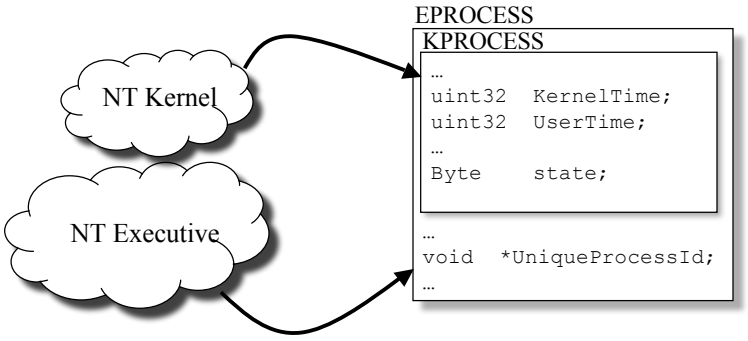
(a) a doubly linked listed with three elements

(b) an empty doubly linked list



Windows NT Process Descriptor

Slide 6-24



NT Kernel handles object management, interrupt handing, thread scheduling

NT Executive handles all other aspect of a process

Windows NT Process Descriptor (2)

Slide 6-25

- Kernel process object includes:
  - Pointer to the page directory
  - Kernel & user time
  - Process base priority
  - Process state
  - List of the Kernel thread descriptors that are using this process

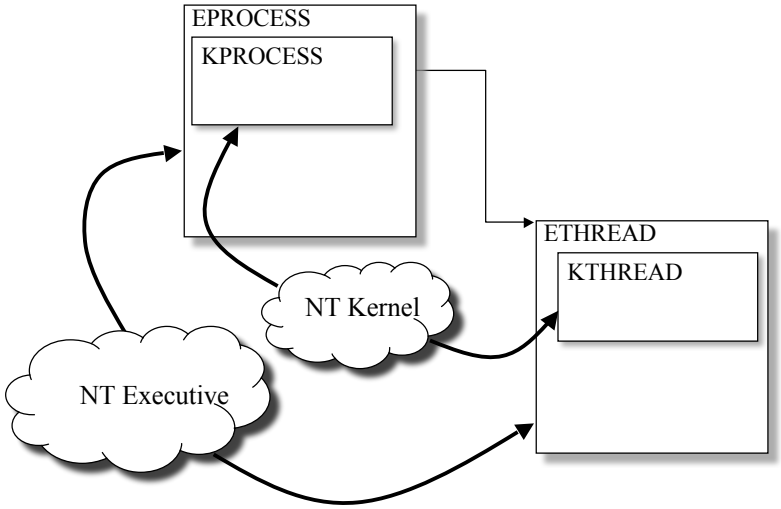
Windows NT Process Descriptor (3)

Slide 6-26

- Parent identification
- Exit status
- Creation and termination times.
- Memory status
- Security information
- executable image
- Process priority class used by the thread scheduler.
- A list of handles used by this process
- A pointer to Win32-specific information

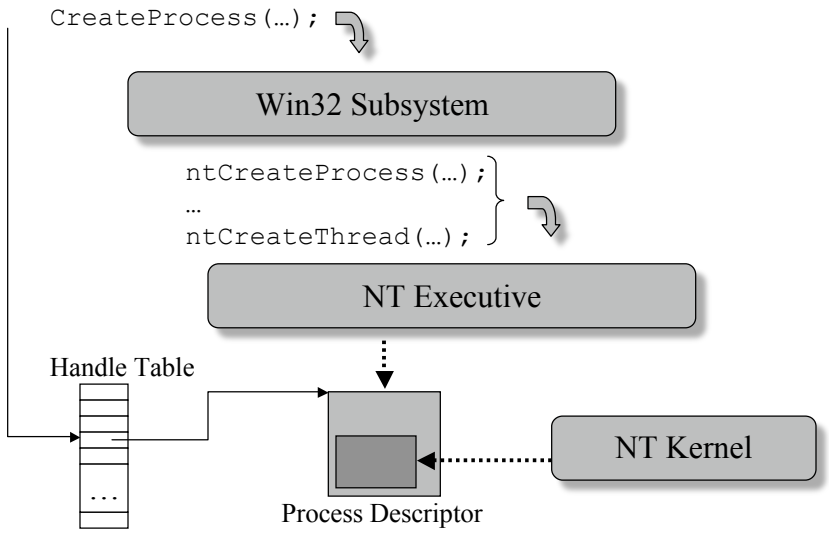
Windows NT Thread Descriptor

Slide 6-27



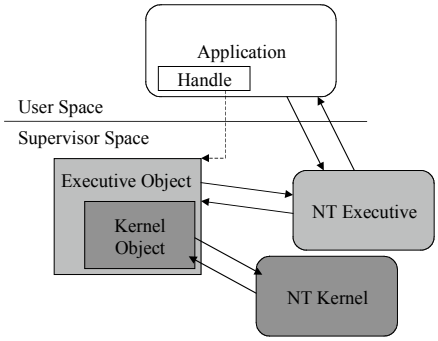
Creating a Process in NT

Slide 6-28



Windows NT Handles

Slide 6-29



Thread Abstraction

Slide 6-30

Process Manager has *algorithms* to control threads and thread descriptor (*data structure*) to keep track of threads.

Management Tasks

- Create/destroy thread
- Allocate thread-specific resources
- Manage thread context switching

Thread Descriptor

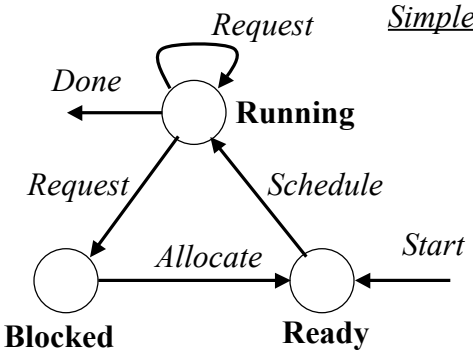
- state
- execution stats
- process (reference to associated process)
- list of related threads
- stack (reference to stack)
- thread-specific resources

State of a Process/Thread

Slide 6-31

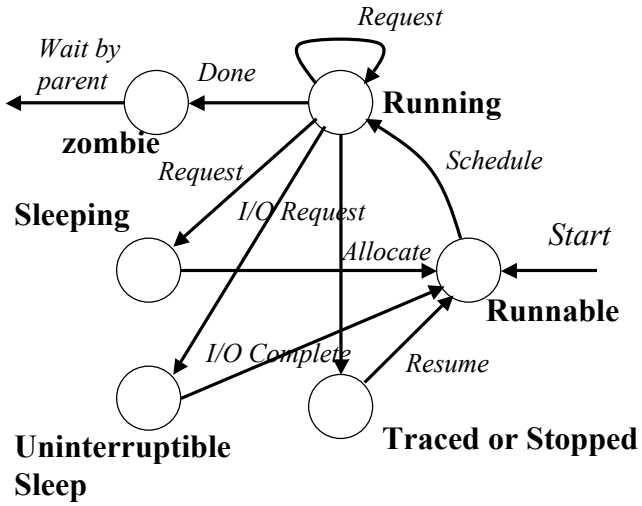
State Variable - summary status of the process/thread which is located in descriptor

Simple State Diagram



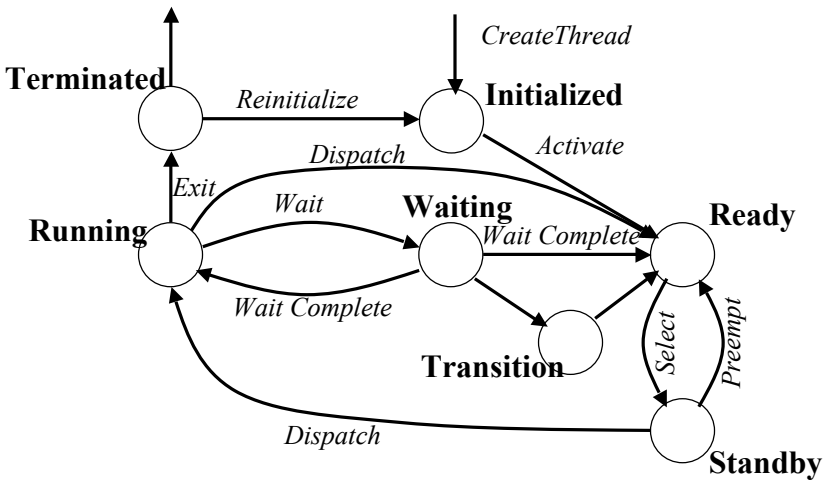
UNIX State Transition Diagram

Slide 6-32



Slide 6-33

Windows NT Thread States



Slide 6-34

Resources

Resource: Anything that a process can request and then become blocked because that thing is not available.

Resource Descriptors

- Internal resource name
- Total Units
- Available Units
- List of available units
- List of Blocked processes

Resources

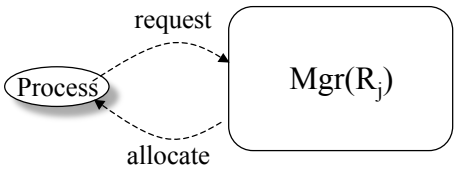
$R = \{R_j \mid 0 \le j < m\}$ = resource types
$C = \{c_j \ge 0 \mid \forall\ R_j \in R\ (0 \le j < m)\}$ = units of R_j available

Reusable resource: After a unit of the resource has been allocated, it must ultimately be released back to the system. E.g., CPU, primary memory, disk space, … The maximum value for c_j is the number of units of that resource

Consumable resource: There is no need to release a resource after it has been acquired. E.g., a message, input data, … Notice that c_j is unbounded.

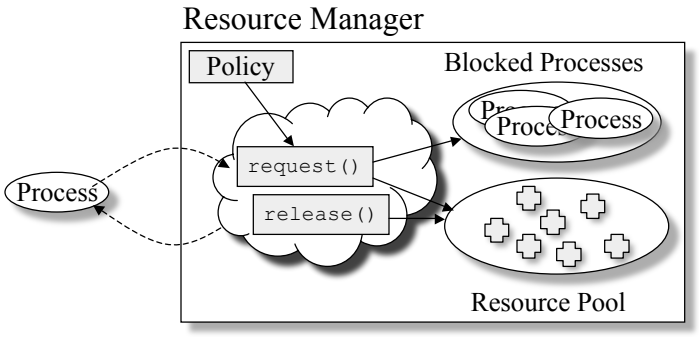
Using the Model

- There is a resource manager, $Mgr(R_j)$ for every R_j
- Process p_i can *request* units of R_j if it is currently *running*
  - p_i can only request $n_i \le c_j$ units of reusable R_j
  - p_i can request unbounded # of units of consumable R_j
- $Mgr(R_j)$ can allocate units of R_j to p_i



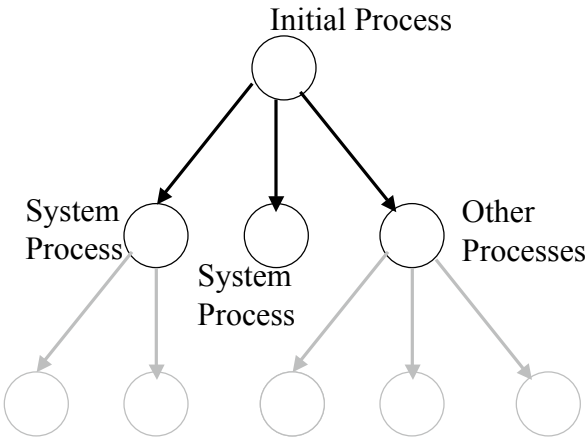
A Generic Resource Manager

Slide 6-37



Process Hierarchies

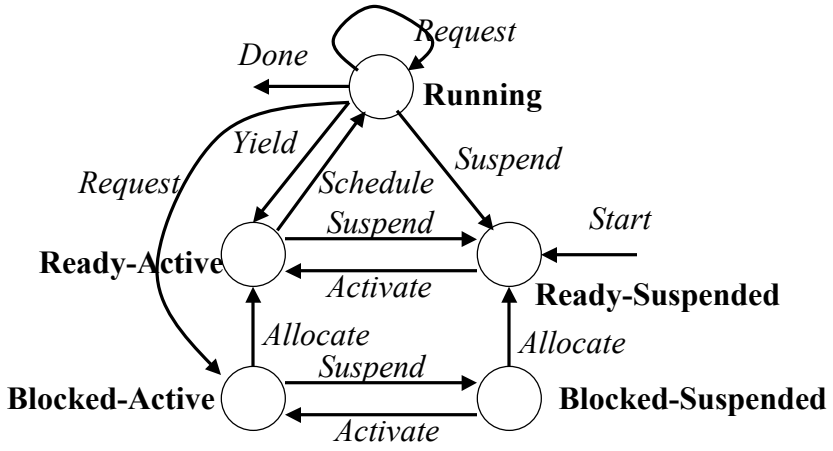
Slide 6-38



Process Hierarchies

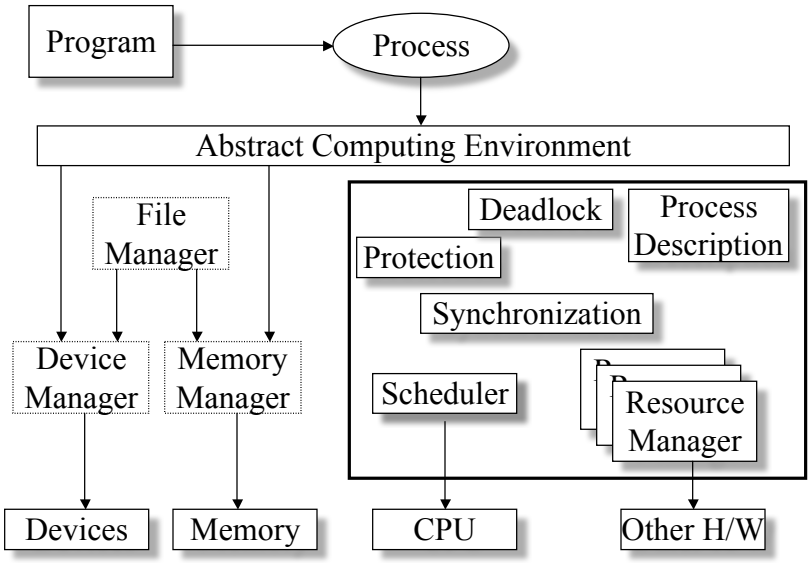
Slide 6-39

- Parent-child relationship may be significant: parent controls children's execution



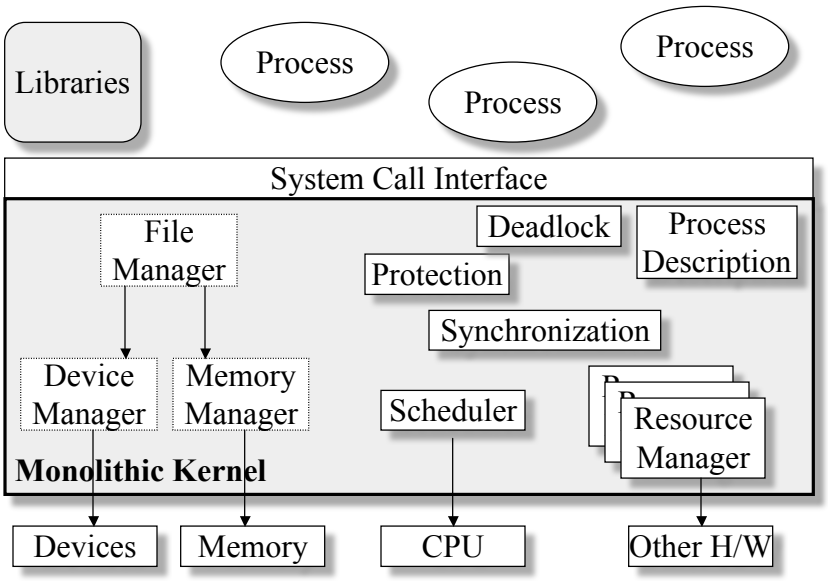
Process Manager Overview

Slide 6-40



Slide 6-41

UNIX Organization



Slide 6-42

Windows NT Organization