Process Description and Control

Chapter 2

1

1

Learning Objectives

At the end of the chapter, the students are able to:

- Understand the concept of process and process management
- Understand the various features of processes, including scheduling, creation and termination, and communication
- Understand the notion of a thread – a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems.

2

2

Process

- A process is a program currently executing.
  - An instance of a program running on a computer
  - It is a unit of work within the system.
  - Program is a *passive entity*, process is an *active entity*.
  - The entity that can be assigned to and executed on a processor
  - A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system instructions

3

3

Process Management

- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- A process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

4

4

Process Elements

1. Identifier – unique to the process
2. State – executing/running or not running
3. Priority – relative to other processes
4. Program counter – address of next instruction in the process
5. Memory pointers – to program code and data
6. Context data – data in the registers while process is executing
7. I/O status information – outstanding I/O requests, I/O devices assigned to the process, files used.
8. Accounting information – amount of processor time & clock time used, time limits, account numbers, etc.

5

5

Process Control Block

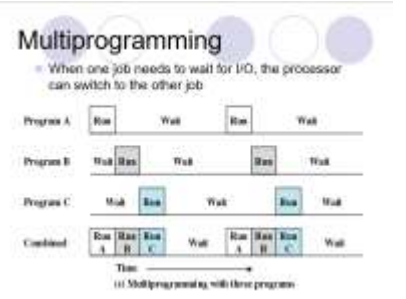
- Contains the process elements
- One of the fundamental data structures in an OS
- Created and managed by the operating system
- Allows support for multiple processes
  - it holds all the information needed to suspend a running process so that when the process enters the run state later the execution context can be restored to a point where the interruption is completely transparent to the process.

6

6

Trace of Process

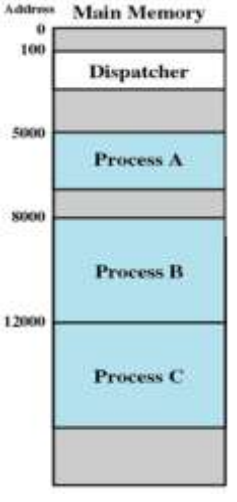
- Sequence of instruction that execute for a process
- Dispatcher switches the processor from one process to another



16

16

Trace of Processes



Snapshot of example execution

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004	I/O operation	12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011
(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C

5000 = starting address of program of Process A, with 12 instructions
8000 = starting address of program of Process B, with 4 instructions, then perform I/O operation
12000 = starting address of program of Process C, with 12 instructions

17

17

1	5000	
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
	Time out	
7	100	
8	101	
9	102	Dispatcher
10	103	
11	104	
12	105	
13	8000	
14	8001	
15	8002	
16	8003	
	I/O request	
17	100	
18	101	
19	102	
20	103	Dispatcher
21	104	
22	105	
23	12000	
24	12001	
25	12002	
26	12003	
27	12004	
28	12005	
	Time out	
29	100	
30	101	
31	102	Dispatcher
32	103	
33	104	
34	105	
35	5006	
36	5007	
37	5008	
38	5009	
39	5010	
40	5011	
	Time out	
41	100	
42	101	
43	102	Dispatcher
44	103	
45	104	
46	105	
47	12006	
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
	Time out	

Assume OS only allows execution of 6 instructions before interrupt

100 = starting address of dispatcher program, with 6 instructions

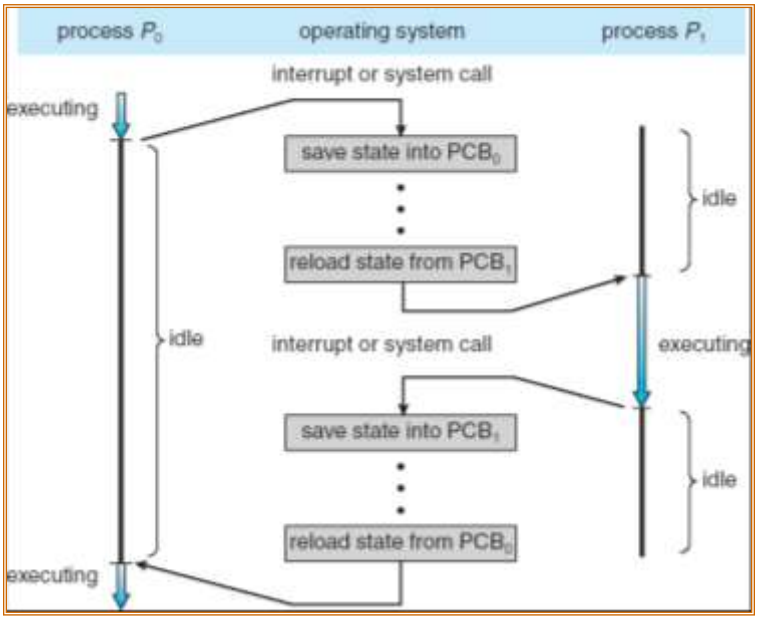
shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.4 Combined Trace of Processes of Figure 3.2

18

18

CPU Switch From Process to Process



19

19

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

20

20

Change of Process State

- Save context of processor including program counter and other registers
- Update the process control block of the process that is currently in the Running state
- Move process control block to appropriate queue – ready; blocked; ready/suspend
- Select another process for execution
- Update the process control block of the process selected
- Update memory-management data structures
- Restore context of the selected process

21

21

When to Switch a Process

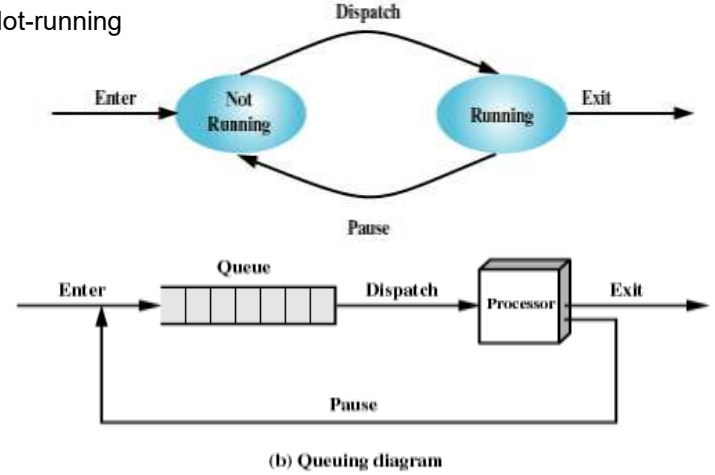
- Clock interrupt
  - process has executed for the maximum allowable time slice
- I/O interrupt
- Memory fault
  - memory address is in virtual memory so it must be brought into main memory
- Trap
  - error or exception occurred
  - may cause process to be moved to Exit state
- Supervisor call
  - such as file open

22

22

Two-State Process Model

- Process may be in one of two states
  - Running
  - Not-running

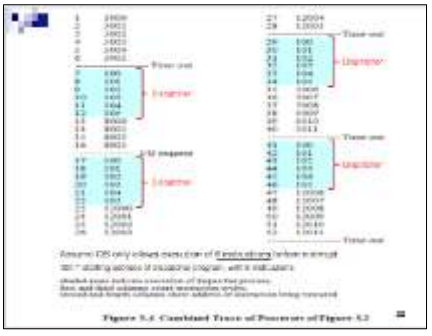


(b) Queuing diagram

23

23

Processes



- Not-running
  - Ready to execute
  - Blocked - waiting for I/O
- Dispatcher cannot just select the process that has been in the queue the longest because it may be blocked

24

24

A Five-State Model

- Running
- Ready
- Blocked
- New
- Exit

Not-running

25

25

A Five-State Model

- **Running** – currently executing
- **Ready** – prepared to execute
- **Blocked** – cannot execute until some event occurs, e.g. I/O operation
- **New** – just created, typically not loaded into MM
- **Exit** – released from the pool of executable processes, either halt (completed) or aborted.

26

26

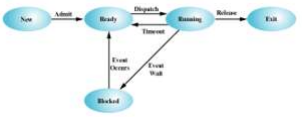
Five-State Process Model

Figure 3.6 Five-State Process Model

27

27

A Five-State Model



New – just created, typically not loaded into MM

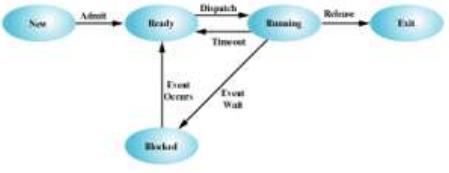
- The OS must:
  - build PCB for the process
  - allocate necessary resources, e.g. memory to hold the program
  - locate program executable file & any initial data needed by the process
  - execute appropriate routines to load initial parts of the process into memory.

28

A Five-State Model

Ready – prepared to execute

- Even if process is in ready state, it does not start executing until the OS gives it control of the CPU
- If there are >1 process in the ready state, the CPU scheduler or process scheduler chooses one to execute (details in next chapter)

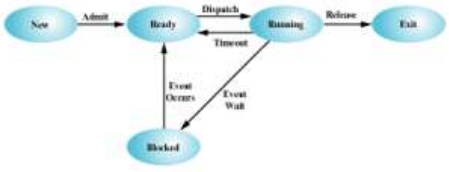


29

A Five-State Model

Running – currently executing

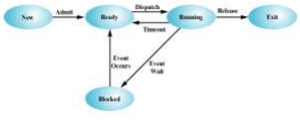
- Process gains control of the CPU
- In single processor system, only one process can get control of the CPU at one time.



30

30

A Five-State Model



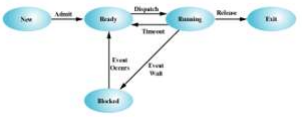
Blocked / waiting – cannot execute until some event occurs, e.g. I/O operation

- If the process requires some resources that is not available; or if it needs some I/O to occur (e.g. waiting for a keystroke or reading from a file) before it can continue processing.
- A process remains in the block state until the resource it needs is allocated to it, or its I/O request is completed → then it is moved to the ready state.

31

31

A Five-State Model

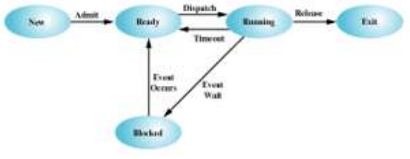


Exit / terminate – released from the pool of executable processes, either halt (completed) or aborted.

- When a process reaching its end (finishes) or having a fatal error (or exception that causes the OS to abort it)
- OS will do the cleanup operations on the process, e.g. deleted the PCB and data structures, and free up the process memory and other resources

32

Five State Model Transition



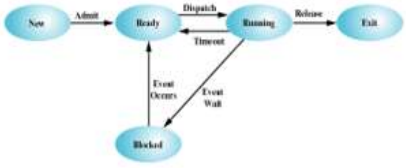
NULL----NEW: A new process is created due to any of the four reasons described in the creation of processes (New batch Job, Interactive logon, To provide service & Spawning).

NEW-----READY: Operating system moves a process from new state to ready state, when it is prepared to accept an additional process. There could be limit on number of processes to be admitted to the ready state.

READY---RUNNING: Any process can be moved from ready to running state when ever it is scheduled.

33

Five State Model Transition



RUNNING----EXIT: The currently running process is terminated if it has signaled its completion or it is aborted.

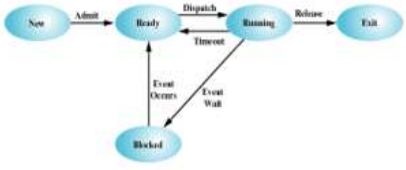
RUNNING----READY:

The most commonly known situation is that currently running process has taken its share of time for execution (Time out). Also, in some events a process may have to be admitted from running to ready if a high priority process has occurred.

34

34

Five State Model Transition



RUNNING----BLOCKED:

A process is moved to the blocked state, if it has requested some data for which it may have to wait. For example the process may have requested a resource such as data file or shared data from virtual memory, which is not ready at that time.

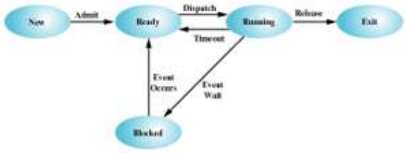
BLOCKED---READY:

A process is moved to the ready state, if the event for which it is waiting has occurred.

35

35

Five State Model Transition



There are two more transition but are not shown for clarity.

READY----EXIT:

This is the case for example a parent process has generated a single or multiple children processes & they are in the ready state. Now during the execution of the process it may terminate any child process, therefore, it will directly go to exit state.

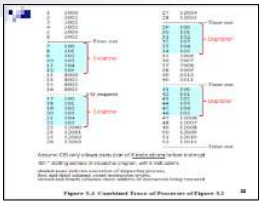
BLOCKED----EXIT:

Similarly as above, during the execution of a parent process any child process blocked/waiting for an event to occur may directly go to exit if the parent itself terminates.

36

36

Process States



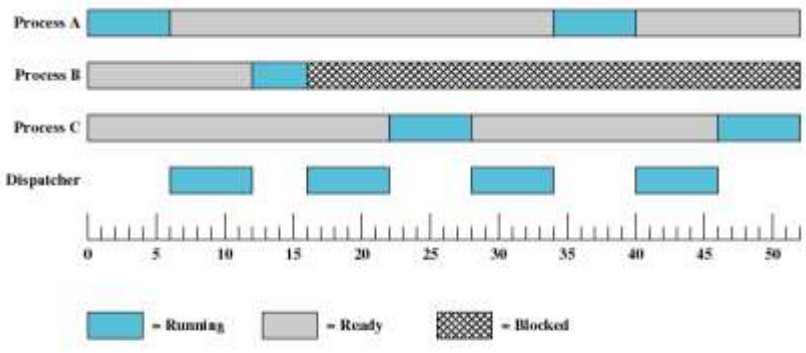
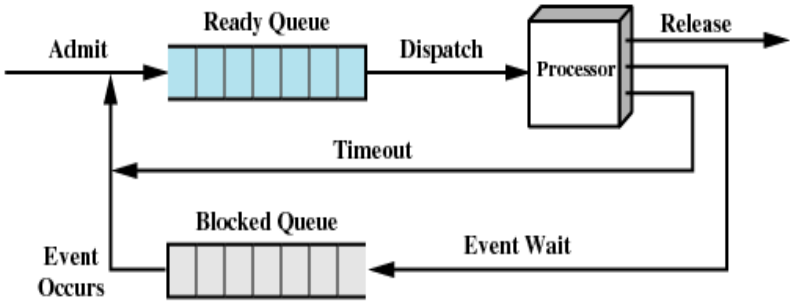


Figure 3.7 Process States for Trace of Figure 3.4

37

37

Using Two Queues

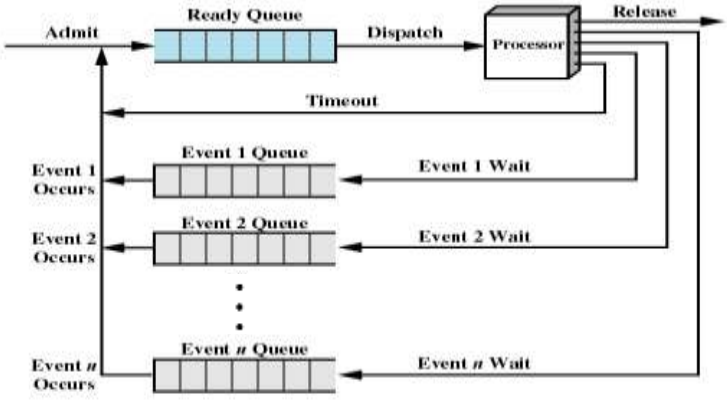


(a) Single blocked queue

38

38

Multiple Blocked Queues



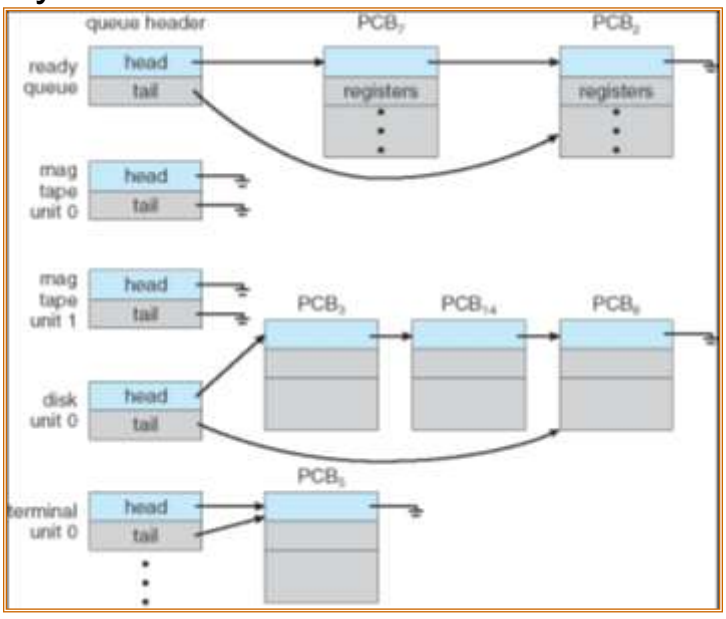
(b) Multiple blocked queues

Figure 3.8 Queuing Model for Figure 3.6

39

39

Ready Queue And Various I/O Device Queues

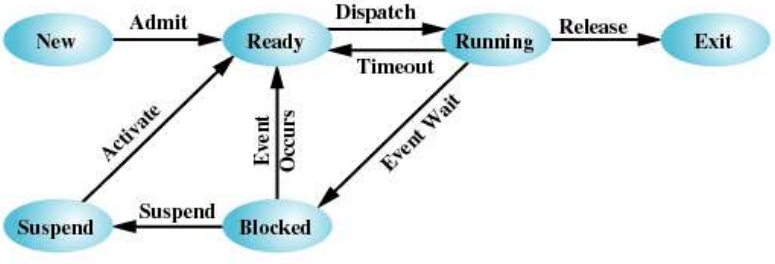


40

40

Suspended Processes

- Processor is faster than I/O so all processes could be waiting for I/O
- Swap these processes to disk to free up more memory
- Blocked state becomes suspend state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend



(a) With One Suspend State

41

41

Reasons for Process Suspension

Swapping	The operating system needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The operating system may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.
Parent process request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendents.

42

42

Processes and Resources

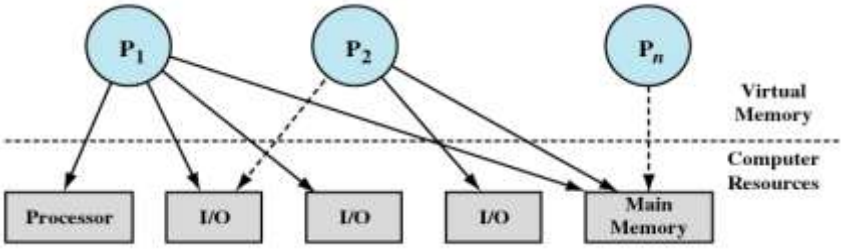
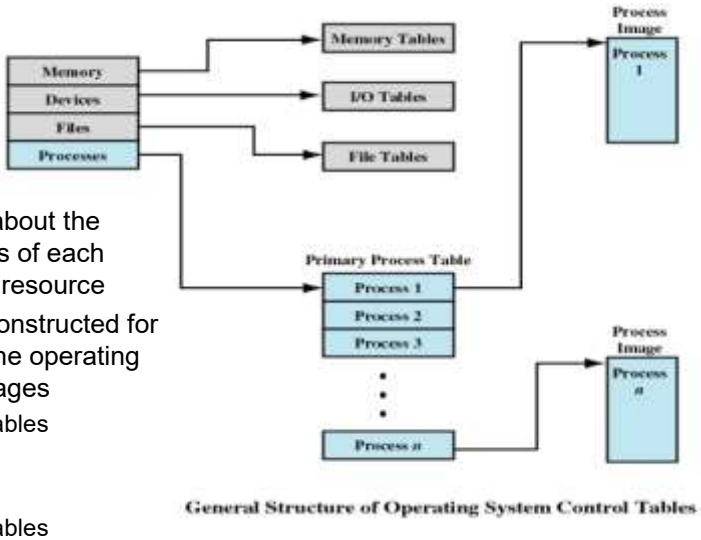


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

43

43

OS Control Structures



General Structure of Operating System Control Tables

- Information about the current status of each process and resource
- Tables are constructed for each entity the operating system manages
  - Memory tables
  - I/O tables
  - File tables
  - Process tables

44

44

Memory Tables

Includes:

- Allocation of main memory to processes
- Allocation of secondary memory to processes
- Protection attributes for access to shared memory regions
- Information needed to manage virtual memory

File Tables

- Existence of files
- Location on secondary memory
- Current status
- Attributes
- Sometimes this information is maintained by a file management system

I/O Tables

- I/O device is available or assigned
- Status of I/O operation
- Location in main memory being used as the source or destination of the I/O transfer

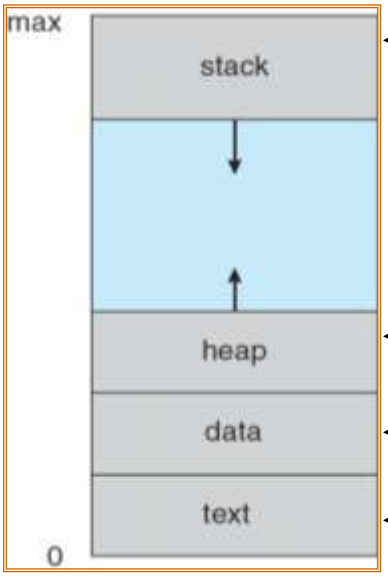
Process Tables

- Where process is located
- Attributes in the process control block
  - Program
  - Data
  - Stack

45

45

Process Image (in Memory)



- Contains temporary data (function var, return address, local var)
- (optional) Memory dynamically allocated during process runtime
- Contains global var
- Program code

46

46

Process Image

Table 3.4 Typical Elements of a Process Image

User Data
The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified.

User Program
The program to be executed.

System Stack
Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls.

Process Control Block
Data needed by the operating system to control the process (see Table 3.5).

47

Process Creation

- Assign a unique process identifier
- Allocate space for the process
- Initialize process control block
- Set up appropriate linkages
  - Ex: add new process to linked list used for scheduling queue
- Create or expand other data structures
  - Ex: maintain an accounting file

48

48

Process Creation

Table 3.1 Reasons for Process Creation

New batch job	The operating system is provided with a batch job control stream, usually on tape or disk. When the operating system is prepared to take on new work, it will read the next sequence of job control commands.
Interactive logon	A user at a terminal logs on to the system.
Created by OS to provide a service	The operating system can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).
Spawned by existing process	For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.

49

49

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

50

50

Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

51

51

Process Termination

Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time ("wall clock time"), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.
Memory unavailable	The process requires more memory than the system can provide.
Bounds violation	The process tries to access a memory location that it is not allowed to access.
Protection error	The process attempts to use a resource such as a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.
Arithmetic error	The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.

Reasons for process termination

52

52

Process Termination

Time overrun	The process has waited longer than a specified maximum for a certain event to occur.
I/O failure	An error occurs during input or output, such as inability to find a file, failure to read or write after a specified maximum number of tries (when, for example, a defective area is encountered on a tape), or invalid operation (such as reading from the line printer).
Invalid instruction	The process attempts to execute a nonexistent instruction (often a result of branching into a data area and attempting to execute the data).
Privileged instruction	The process attempts to use an instruction reserved for the operating system.
Data misuse	A piece of data is of the wrong type or is not initialized.
Operator or OS intervention	For some reason, the operator or the operating system has terminated the process (for example, if a deadlock exists).
Parent termination	When a parent terminates, the operating system may automatically terminate all of the offspring of that parent.
Parent request	A parent process typically has the authority to terminate any of its offspring.

Reasons for process termination

53

53

Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

54

54

Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

55

55

Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional, but is usually bi-directional

56

56

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several comm. links
  - Link may be unidirectional or bi-directional

57

57

Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:

  send(*A, message*) – send a message to mailbox A

  receive(*A, message*) – receive a message from mailbox A

- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A
  - P_1, sends; P_2 and P_3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

58

58

Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
  - **Blocking send** has the sender block until the message is received
  - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue
  - **Non-blocking** receive has the receiver receive a valid message or null

59

59

Client-Server Communication

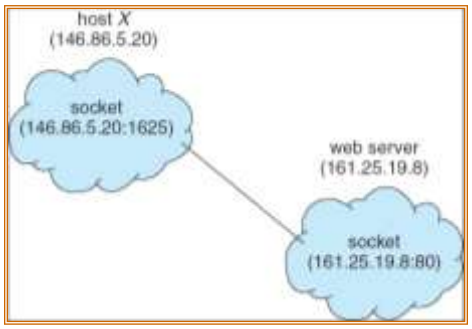
- Sockets
- Remote Procedure Calls
- Remote Method Invocation (Java)

60

60

Socket Communication

- A socket is defined as an *endpoint for communication*
- Concatenation of IP address and port
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets

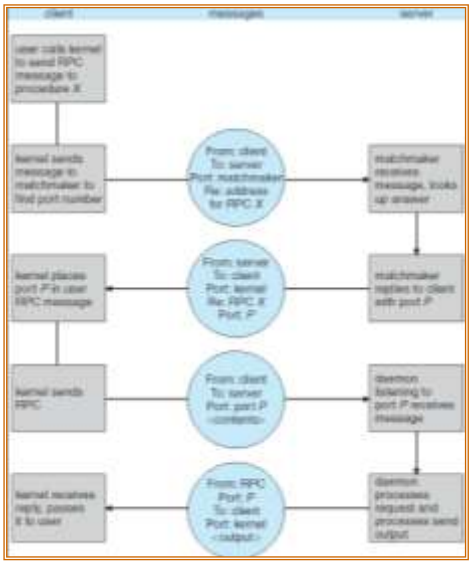


61

61

Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- **Stubs** – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and *marshalls* the parameters.
- The server-side stub receives this message, unpacks the marshalled parameters, and peforms the procedure on the server.



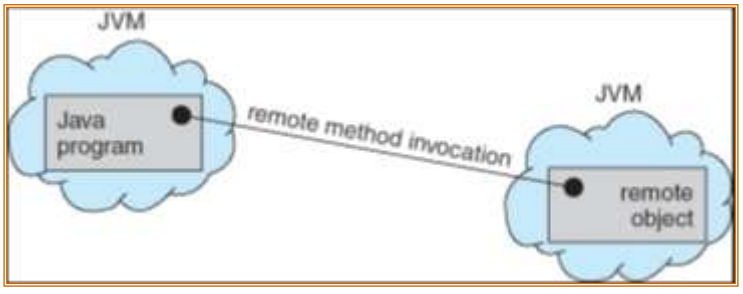
Execution of RPC

62

62

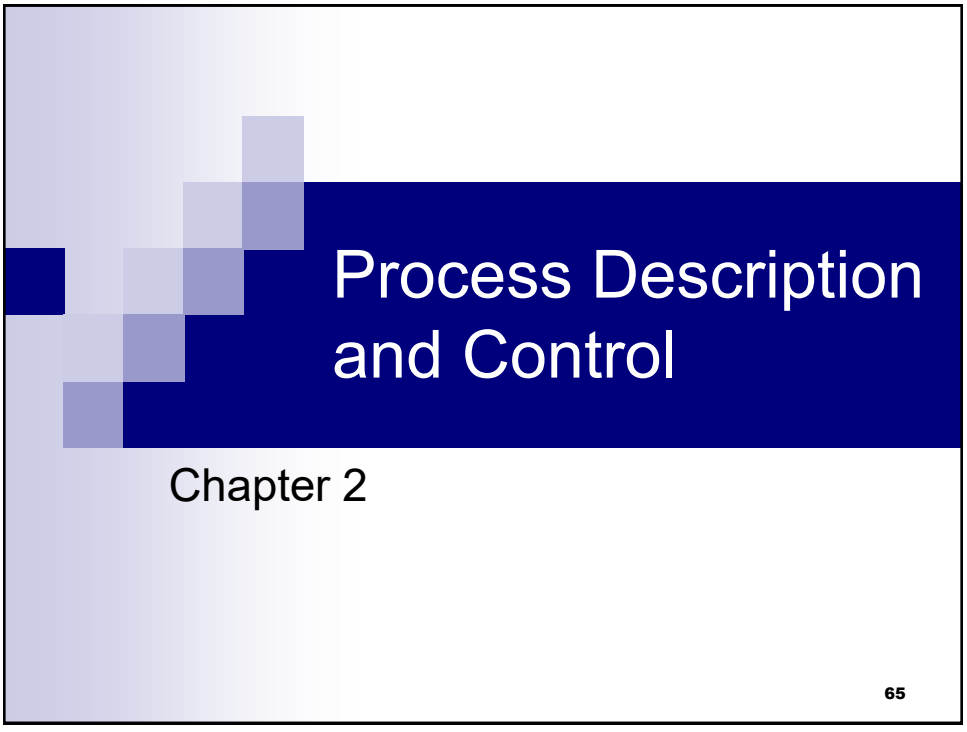
Remote Method Invocation

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
- RMI allows a Java program on one machine to invoke a method on a remote object.

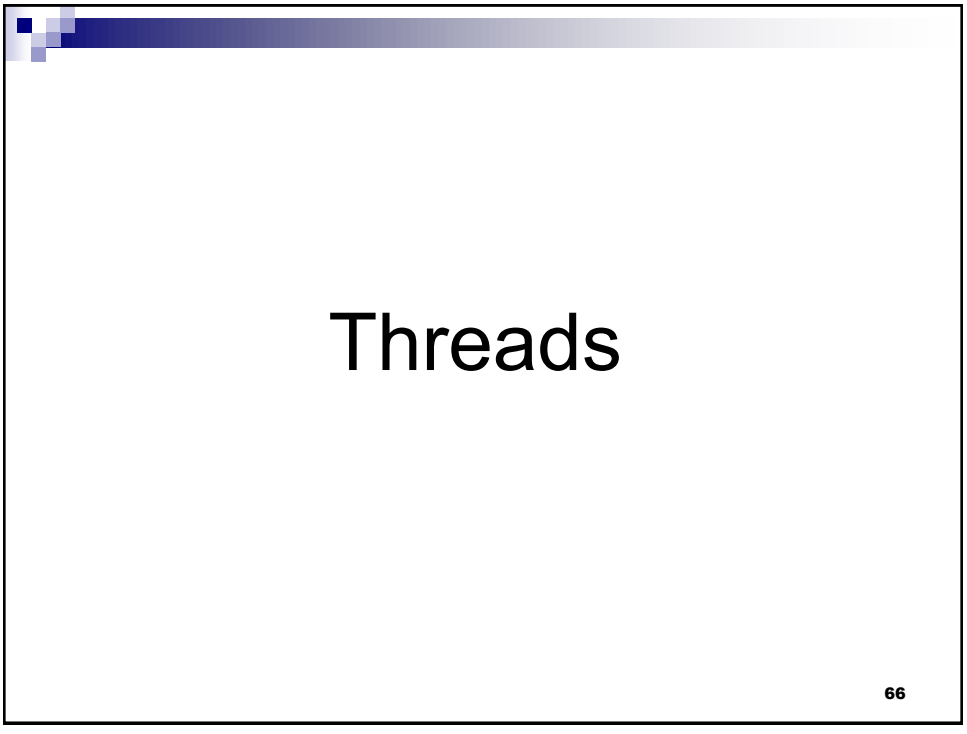


63

63



65



66

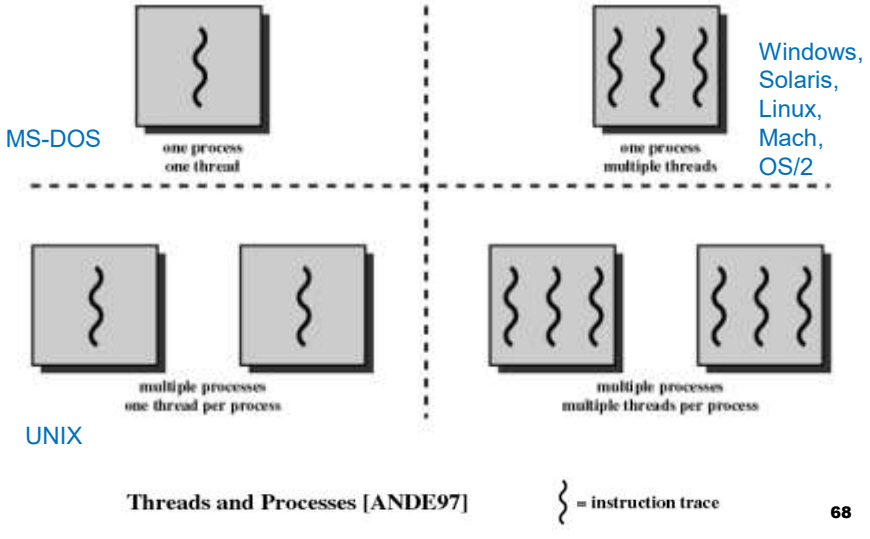
- Concept of process – two characteristics:
  - Resource ownership
    - process includes a virtual address space to hold the process image
  - Scheduling/execution
    - follows an execution path that may be interleaved with other processes
- These two characteristics are treated independently by the operating system
- Dispatching is referred to as a thread or lightweight process
- Resource ownership is referred to as a process or task

67

67

Multithreading

Operating system supports multiple threads of execution within a single process



68

68

Process Management

- Process have a virtual address space which holds the process image
- Protected access to processors, other processes, files, and I/O resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread

69

69

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

70

70

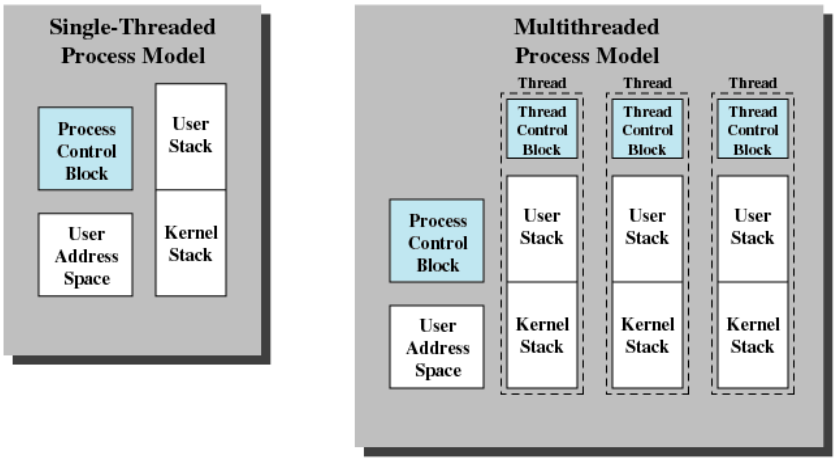


Figure 4.2 Single Threaded and Multithreaded Process Models

71

71

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

72

72

Uses of Threads in a Single-User Multiprocessing System

- Foreground to background work
  - E.g: A spreadsheet program:
    - one thread display menus and read user input
    - while another thread executes user commands and updates the spreadsheet
- Asynchronous processing
  - E.g: protection against power failure:
    - One thread writes the RAM buffer to disk every minute – for periodic backup and schedules itself directly with the OS

73

73

Uses of Threads in a Single-User Multiprocessing System

- Speed of execution
  - Compute one batch of data while reading the next batch from a device
  - On multiprocessor system, multiple threads may execute simultaneously. Thus, if one thread is blocked for an I/O, another thread may be executing.
- Modular program structure
  - Easier to design and implement (using threads) programs involving a variety of activities or a variety of sources/destinations of I/O.

74

74

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

75

75

Thread States

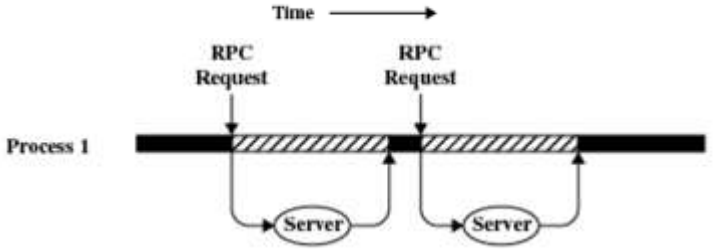
- States associated with a change in thread state
  - Spawn
    - Spawn another thread
  - Block
  - Unblock
  - Finish
    - Deallocate register context and stacks

76

76

Example: Remote Procedure Call using Single Thread

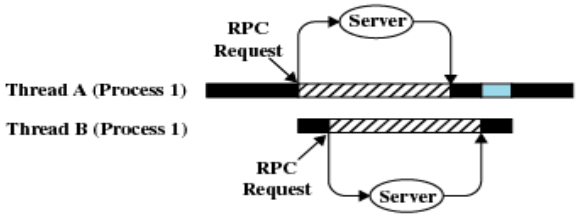
- Assume a program needs to make 2 RPCs to 2 different hosts to get the combined results.
- Using single thread:
  - The result is obtained in sequence
  - The program has to wait for response from each host/server in turn.



77

77

RPC using Two Threads



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

Blocked, waiting for response to RPC
Blocked, waiting for processor, which is in use by Thread B
Running

Figure 4.3 Remote Procedure Call (RPC) Using Threads

78

78

Multithreading

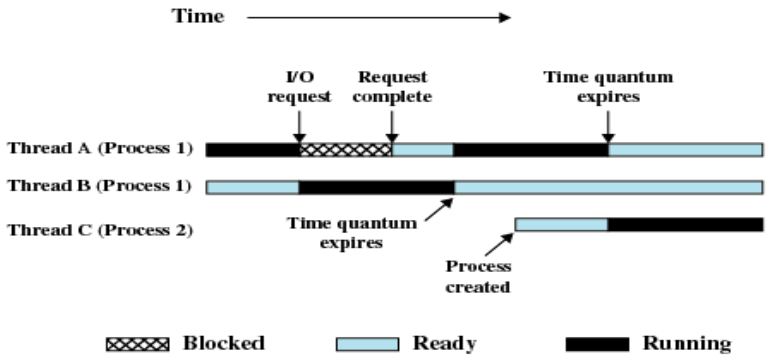
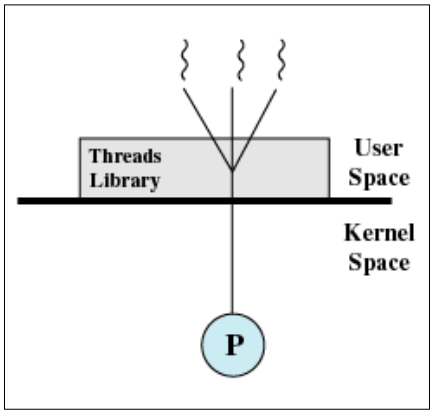


Figure 4.4 Multithreading Example on a Uniprocessor

79

79

User-Level Threads

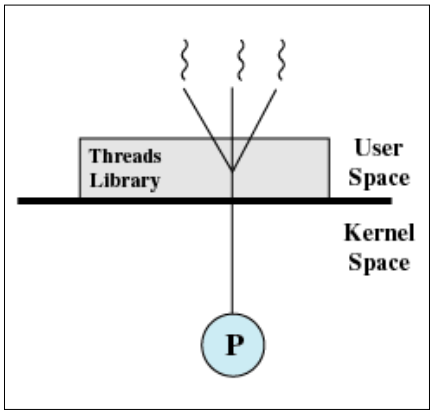


- All thread management is done by the application
- The kernel is not aware of the existence of threads
  - The kernel knows nothing about user-level threads and manages them as if they were single-threaded processes.
- Managed entirely by the run-time system (user-level library).

80

80

User-Level Threads

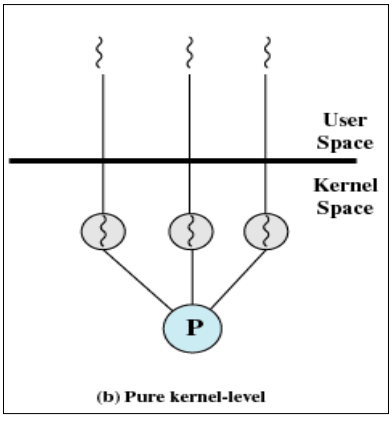


- User-Level threads are small and fast, each thread is represented by a PC, register, stack, and small thread control block.
- Threads library contains codes for:
  - creating and destroying threads
  - passing messages and data between threads
  - scheduling thread execution
  - saving and restoring thread context

81

81

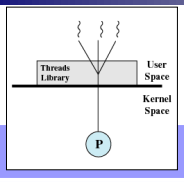
Kernel-Level Threads



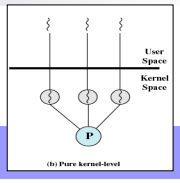
- Windows is an example of this approach
- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis
- Managed by OS.
- All modern OSs support the threading model.
- Implementation of a thread will change according to the OS.

82

82



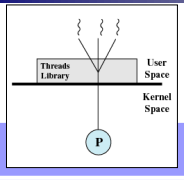
ULT vs. KLT



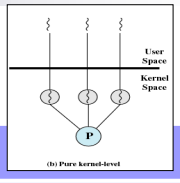
ULT	KLT
Thread switching do not require kernel mode switching/privileges – less overhead.	Transfer of control from one thread to another thread within the same process requires mode switching.
All thread management is done by the application. Thread scheduling is application specific, without disturbing the underlying OS scheduler.	Scheduling is done by the kernel on a thread basis.
The kernel is not aware of the existence of threads. Managed by the user.	The kernel is aware of kernel threads. Kernel maintains context information for the process and the threads.

83

83



ULT vs. KLT

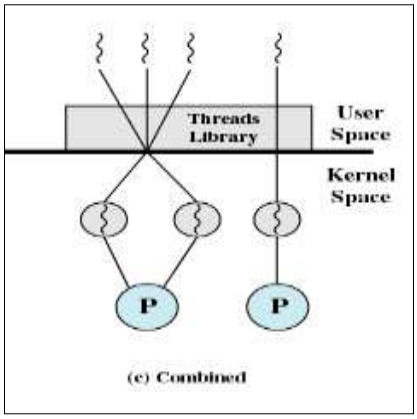


ULT	KLT
If one thread blocks, then the whole process blocks.	If one thread of a process is blocked it can schedule another thread of the same process.
Cannot take advantage of multiprocessing system: one process to one processor at a time, i.e. one thread running at a time.	The kernel can simultaneously schedule threads on different processors.
Executes user-space code, but can still call into kernel space at any time.	Only runs kernel code, not associated with user-space processes.
ULTs can run on any OS – no change needed to the underlying kernel to support ULT – use thread library.	Managed by OS. All modern OSs support the threading model. Implementation of a thread will change according to the OS.

84

84

Combined Approaches



- Example is Solaris
- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads within application
- Advantages:
  - Multiple threads within the same application can run concurrently on a number of processors.
  - Blocking system calls need not block the entire process.

85

85

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

86

86

Pthreads

- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

87

87

Windows XP Threads

- Implements the one-to-one mapping
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the **context** of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

88

88

Linux Threads

- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through **clone()** system call
- **clone()** allows a child task to share the address space of the parent task (process)

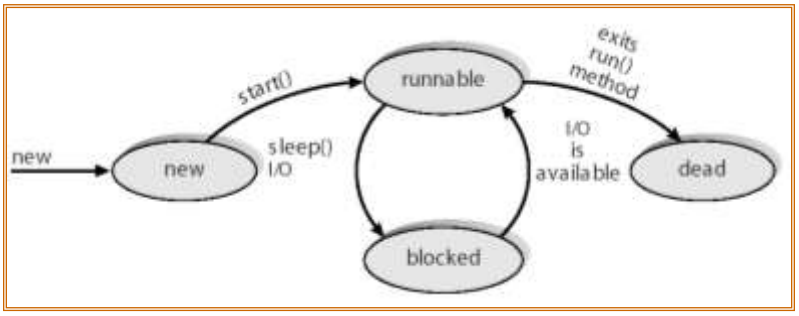
89

89

Java Threads

- Java threads are managed by the JVM
- Java threads may be created by:
  - Extending Thread class
  - Implementing the Runnable interface

Java Thread States



90

90