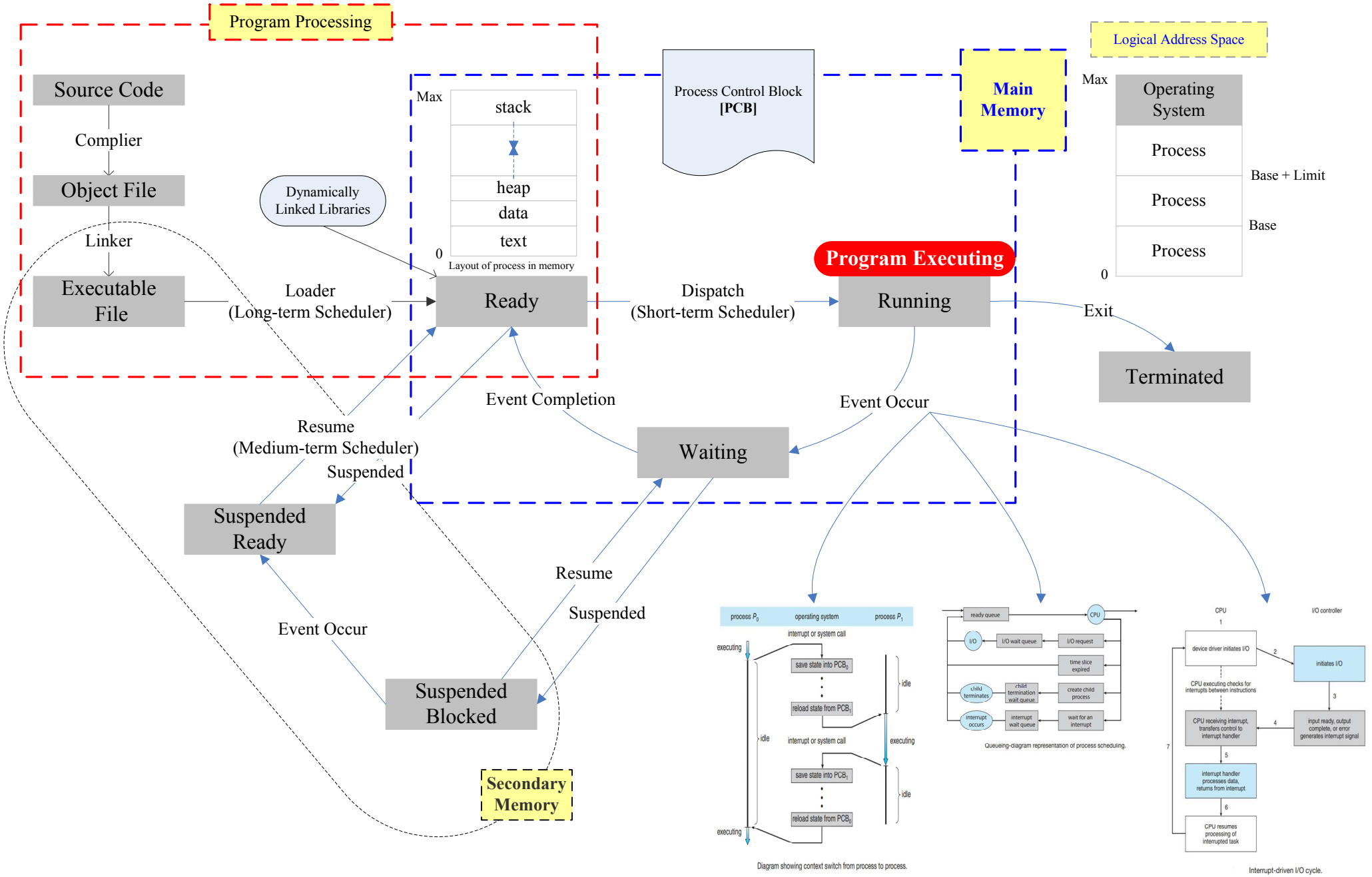
Introduction to Operating System _ Process Management



Introduction to Operating System _ Program Executing _ ALL

Program Processing

Source Code → Complier → Object File → Linker → Executable File → Loader (Long-term Scheduler) → Program in memory

Dynamically Linked Libraries → Linker

Program Running

Process Operation

The processor generates virtual addresses while the memory is accessed using physical addresses. Both the virtual memory and the physical memory are broken into pages, so that a virtual page is mapped to a physical page. Of course, it is also possible for a virtual page to be absent from main memory and not be mapped to a physical address; in that case, the page resides on disk. Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code.

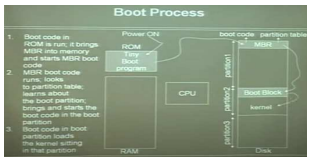
Secondary Memory

- Block Control Block
- File Control Block
- File Allocate Table
- Open File Table
- Program Block
- Program Block
- Swap Space Block
- **Boot Block** Bootstrap Program

BOIS/UFEI Bootstrap Loader

Initial Devices → Devices

Load entire operating system & Start the OS running



Paging

Logic Address: p (Page Number) | d (Page Offset)

Physical Address: f (Frame Number) | d (Frame Offset)

TLB: p | f

TLB hit

TLB miss

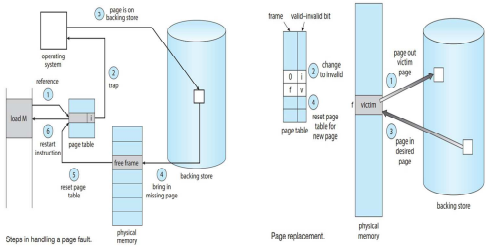
Translation look-aside buffer

Page Fault

Free Frame

YES → Swap

NO → Replace



Virtual View

Page

Main Memory

- Process Control Block **PCB**
- SegmenTable
- Page Table
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- **Operating System** Kernel System Daemons

CPU

STBR(Page-table base register)

STLR(Page-table length register)

PTBR(Segment-table base register)

PTLR(Segment-table length register)

PC(Program Counter)

IR(Instruction Register)

MAR(Memory Address Register)

MDR(Memory Data Register)

Control Unit

ALU

1, 6, 12

2

3

4

5

7

8

9

10

11

Registers for O32 calling convention

Name	Number	Use	Callee must preserve?
$zero	$0	constant 0	N/A
$at	$1	assembler temporary	No
$v0–$v1	$2–$3	values for function returns and expression evaluation	No
$a0–$a3	$4–$7	function arguments	No
$t0–$t7	$8–$15	temporaries	No
$s0–$s7	$16–$23	saved temporaries	Yes
$t8–$t9	$24–$25	temporaries	No
$k0–$k1	$26–$27	reserved for OS kernel	N/A
$gp	$28	global pointer	Yes (except PIC code)
$sp	$29	stack pointer	Yes
$fp	$30	frame pointer	Yes
$ra	$31	return address	N/A

Introduction to Operating System _ Process Management _ Program Executing

Program Processing

Source Code → Complier → Object File → Linker → Executable File → Loader → Program in memory

Dynamically Linked Libraries → Loader

Long-term Scheduler

Process Operation

Process Operation

The processor generates virtual addresses while the memory is accessed using physical addresses. Both the virtual memory and the physical memory are broken into pages, so that a virtual page is mapped to a physical page. Of course, it is also possible for a virtual page to be absent from main memory and not be mapped to a physical address; in that case, the page resides on disk. Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code.

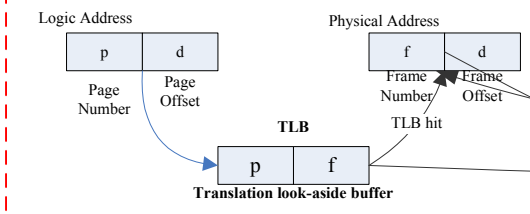
Secondary Memory

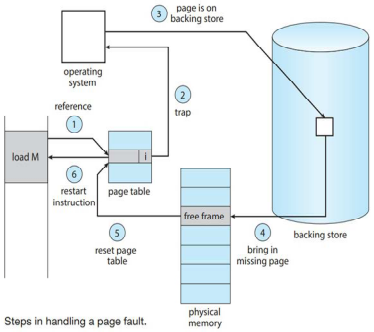
- Block Control Block
- File Control Block
- File Allocate Table
- Open File Table
- Program Block
- Program Block
- Swap Space Block
- **Boot Block** Bootstrap Program

BOIS/UFEI Bootstrap Loader

Initial Devices → Devices

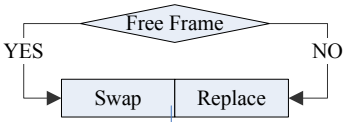
Paging

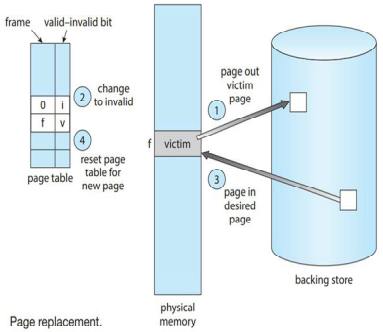




Steps in handling a page fault.

Page Fault



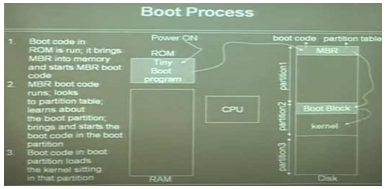


Page replacement.

Page

Virtual View

Load entire operating system
&
Start the OS running



Main Memory

- Process Control Block **PCB**
- SegmenTable
- Page Table
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- **Operating System** Kernel System Daemons

Introduction to Operating System _ Virtual Memory _ ALL

Program Processing

Source Code → Complier → Object File → Linker → Executable File → Loader → Program in memory

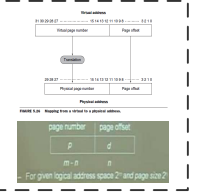
Dynamically Linked Libraries → Linker

Long-term Scheduler

Process Operation

Process Operation

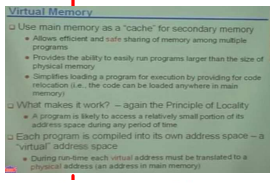
Address generated by CPU is divided into:
- Page number (p) – used as an index into a page table which contains base address of each page in physical memory
- Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit



Virtual Memory

Paging

Secondary Memory



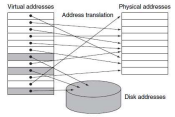


FIGURE 5.23 In virtual memory, blocks of memory (called pages) are mapped from one set of addresses (called virtual addresses) to another set (called physical addresses).

- Block Control Block
- File Control Block
- File Allocate Table
- Open File Table
- Program Block
- Program Block
- Swap Space Block
- **Boot Block**
  Bootstrap Program

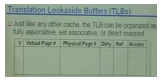
BOIS/UFEI
Bootstrap Loader

Initial Devices

Devices

Logic Address

p	d
Page Number	Page Offset



TLB

p	f

Translation look-aside buffer

TLB hit

TLB miss

Cache

Physical Address

f	d
Frame Number	Frame Offset

Cache miss

TLB miss

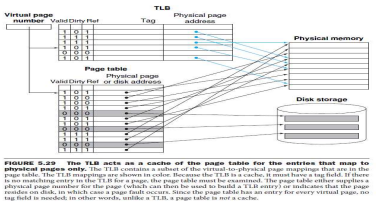
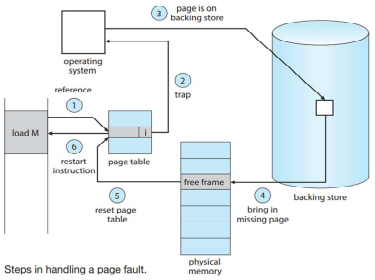


FIGURE 5.29 The TLB acts as a cache of the page table for the entries that map to physical pages only. The TLB contains a subset of the virtual-to-physical page mappings that are in the page table. The TLB mappings are shown in color. Because the TLB is a cache, it must have a tag field. If there is no matching entry in the TLB for a page, the page table must be examined. The page table either supplies a physical page number for the page (which can then be used to build a TLB entry) or indicates that the page resides on disk, in which case a page fault occurs. Since the page table has an entry for every virtual page, no tag field is needed; in other words, unlike a TLB, a page table is not a cache.



Steps in handling a page fault.

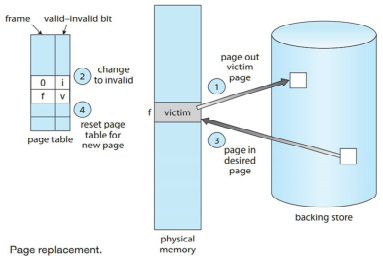
Page Fault

Free Frame

YES

NO

Swap | Replace



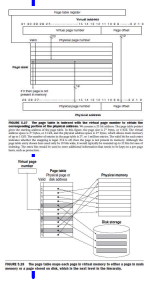
Page replacement.

Page

Virtual View

Main Memory

- Process Control Block **PCB**
- SegmenTable
- Page Table
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- Frame
- **Operating System**
  Kernel
  System Daemons



Load entire operating system
&
Start the OS running

Introduction to Operating System _ Virtual Memory _ Paging

Program Processing

Source Code → Complier → Object File → Linker → Executable File → Loader → Program in memory

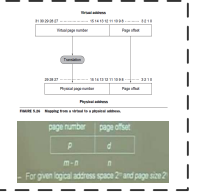
Dynamically Linked Libraries → Linker

Long-term Scheduler

Process Operation

Process Operation

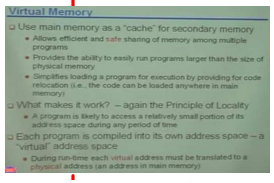
Address generated by CPU is divided into:
- Page number (p) – used as an index into a page table which contains base address of each page in physical memory
- Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit



Virtual Memory

Paging

Secondary Memory



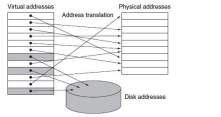


FIGURE 5.29 In virtual memory, blocks of memory (called pages) are mapped from one set of addresses (called virtual addresses) to another set (called physical addresses).

Block Control Block
File Control Block
File Allocate Table
Open File Table
Program Block
Program Block
Swap Space
Boot Block
Bootstrap Program

BOIS/UFEI
Bootstrap Loader

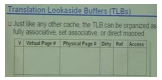
Initial Devices

Devices

Logic Address

p	d

Page Number — Page Offset



TLB

p	f

Translation look-aside buffer

TLB hit

TLB miss

Cache

Physical Address

f	d

Frame Number — Frame Offset

Cache miss

TLB miss

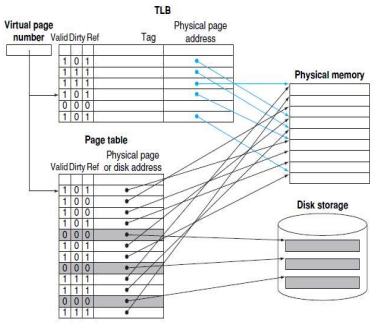


FIGURE 5.29 The TLB acts as a cache of the page table for the entries that map to physical pages only. The TLB contains a subset of the virtual-to-physical page mappings that are in the page table. The TLB mappings are shown in color. Because the TLB is a cache, it must have a tag field. If there is no matching entry in the TLB for a page, the page table must be examined. The page table either supplies a physical page number for the page (which can then be used to build a TLB entry) or indicates that the page resides on disk, in which case a page fault occurs. Since the page table has an entry for every virtual page, no tag field is needed; in other words, unlike a TLB, a page table is *not* a cache.

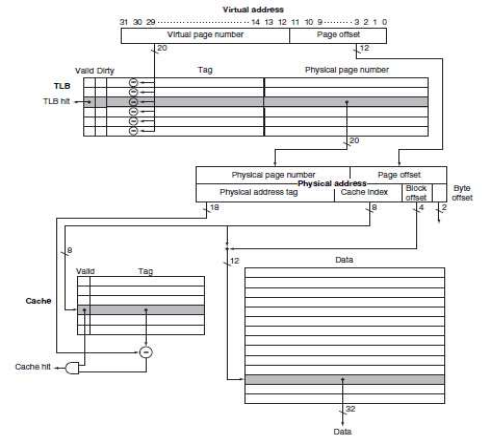
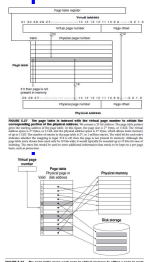


FIGURE 5.30 The TLB and cache implement the process of going from a virtual address to a data item in the intrinsity FastMATH. This figure shows the organization of the TLB and the data cache, assuming a 4 KiB page size. This diagram focuses on a read. Figure 5.31 describes how to handle writes. Note that unlike Figure 5.12, the tag and data RAMs are split. By addressing the long but narrow data RAM with the cache index concatenated with the block offset, we select the desired word in the block without a 16:1 multiplexor. While the cache is direct mapped, the TLB is fully associative. Implementing a fully associative TLB requires that every TLB tag be compared against the virtual page number, since the entry of interest can be anywhere in the TLB. (See content addressable memories in the *Elaboration* on page 408.) If the valid bit of the matching entry is on, the access is a TLB hit, and bits from the physical page number together with bits from the page offset form the index that is used to access the cache.

Main Memory

Process Control Block **PCB**
SegmenTable
Page Table
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Operating System
Kernel
System Daemons



Load entire operating system
&
Start the OS running

Introduction to Operating System _ Virtual Memory _ Page Fault

Program Processing

Source Code → Complier → Object File → Linker → Executable File → Loader → Program in memory

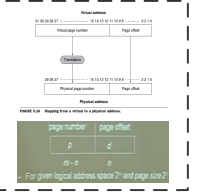
Dynamically Linked Libraries → Linker

Long-term Scheduler

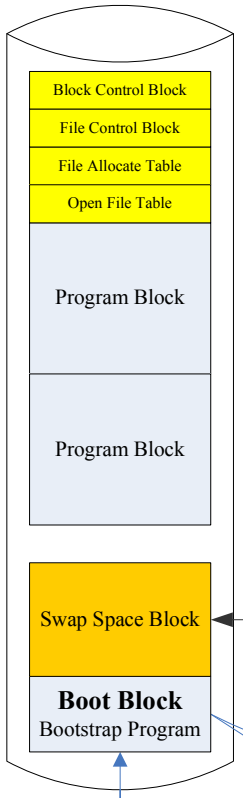
Process Operation

Process Operation

Address generated by CPU is divided into:
- **Page number (p)** – used as an index into a *page table* which contains base address of each page in physical memory
- **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit



Secondary Memory

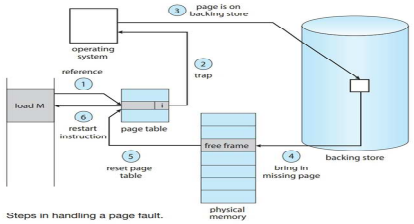


BOIS/UFEI
Bootstrap Loader

Initial Devices

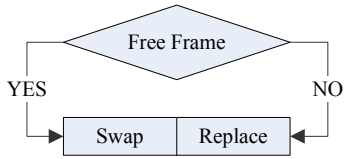
Devices

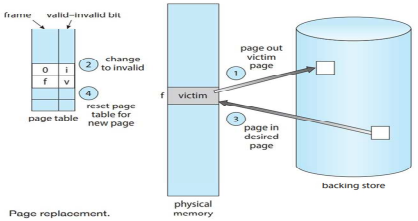
Virtual Memory - Page Fault



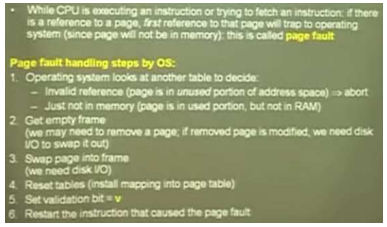
Steps in handling a page fault.

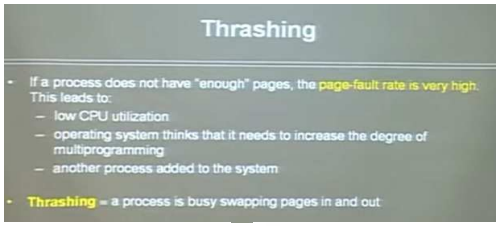
Page Fault

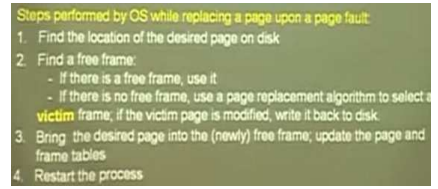


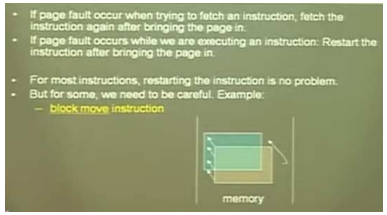


Page replacement.









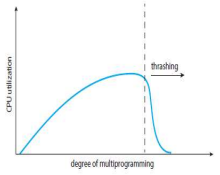


Figure 10.20 Thrashing.

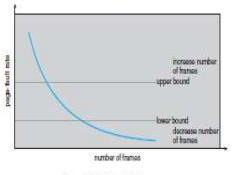
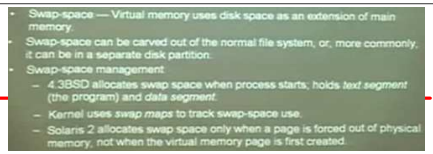


Figure 10.22 Page-fault frequency.

Managing the memory hierarchy between main memory and disk is challenging because of the high cost of page faults. Several techniques are used to reduce the miss rate:

1. Pages are made large to take advantage of spatial locality and to reduce the miss rate.
2. The mapping between virtual addresses and physical addresses, which is implemented with a page table, is made fully associative so that a virtual page can be placed anywhere in main memory.
3. The operating system uses techniques, such as LRU and a reference bit, to choose which pages to replace.

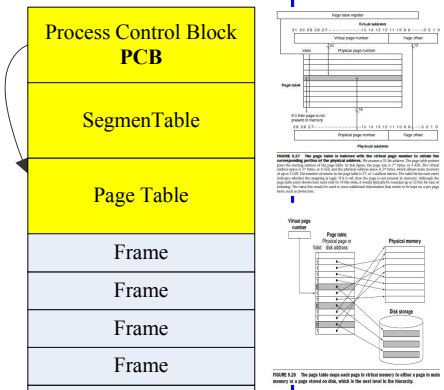


Page

Virtual View

Load entire operating system
&
Start the OS running

Main Memory



Process Control Block
PCB

SegmenTable

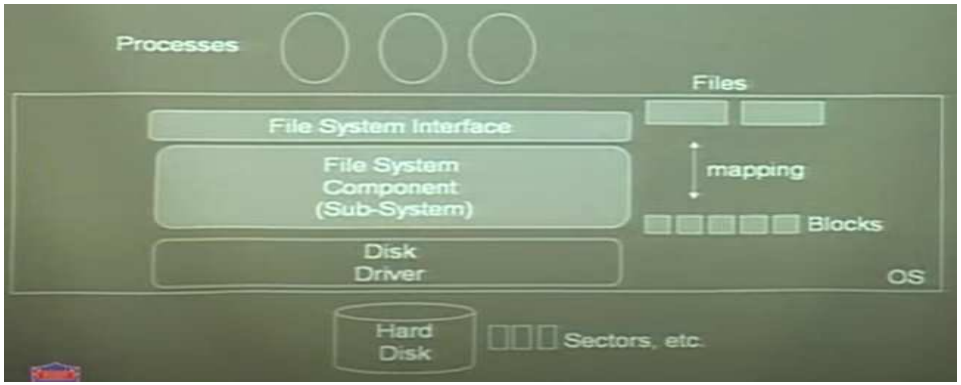
Page Table

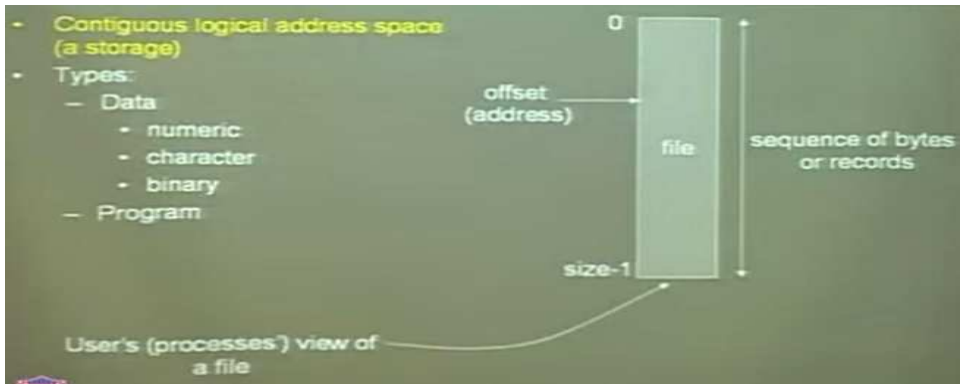
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame
Frame

Operating System
Kernel
System Daemons

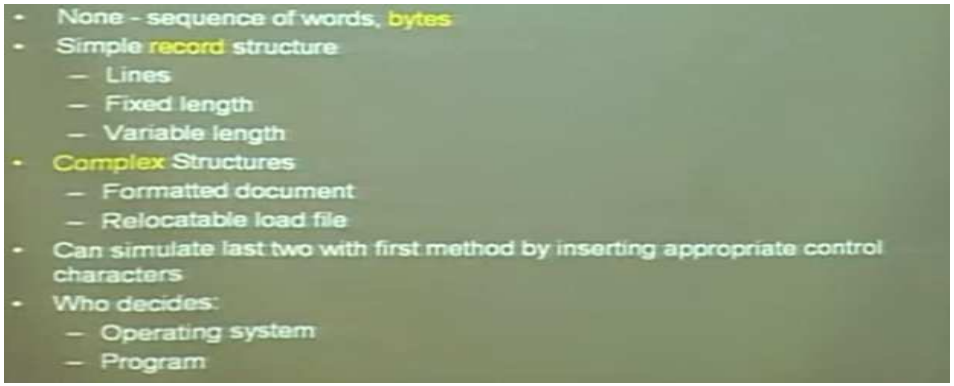
Introduction to Operating System _ File System _ Basic Concept

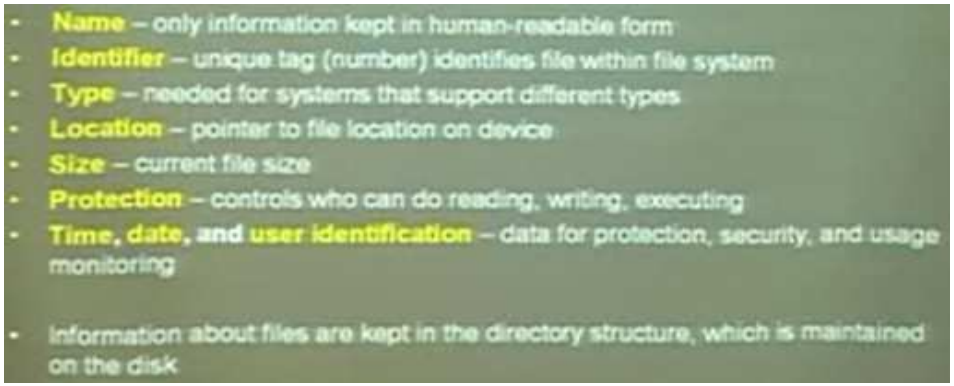
File Concpet

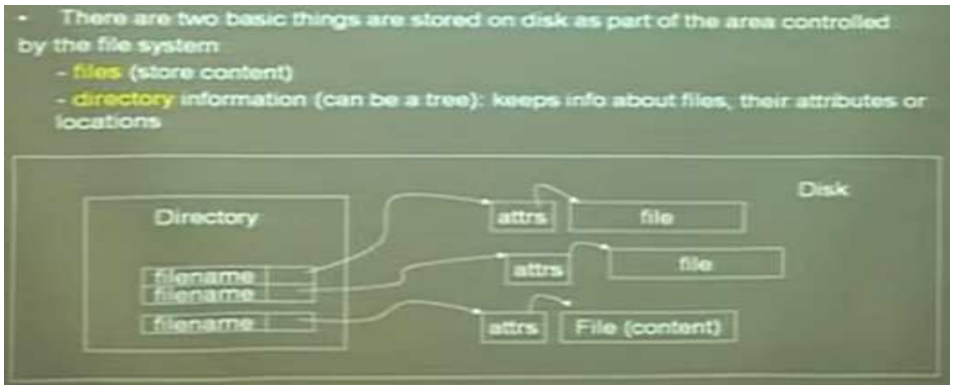




File Structure／Attributes／Directory







Introduction to Operating System _ File System _ File Operations _ Open

File Operations

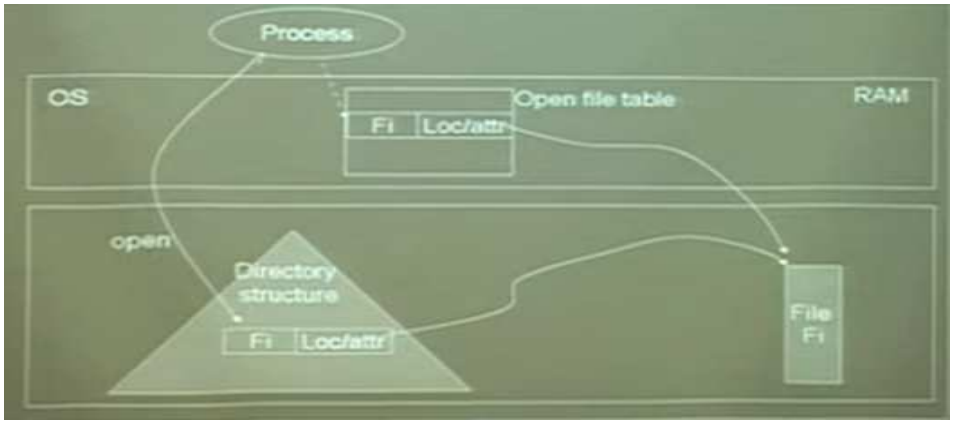
- File is an abstract data type
- Common Operations that are supported by the Operating System:
  - Create
  - Write
  - Read
  - Reposition within file
  - Delete
  - Truncate
- Open(F) – search the directory structure on disk for entry F, and move the content of entry to memory
- Close (F) – move the content of entry F in memory to directory structure on disk

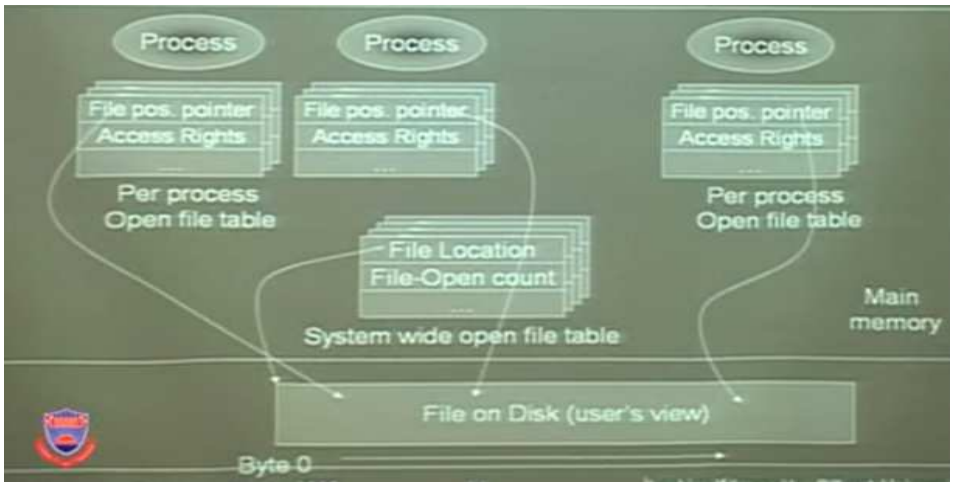
File Types

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Figure 13.3 Common file types.

Open( )

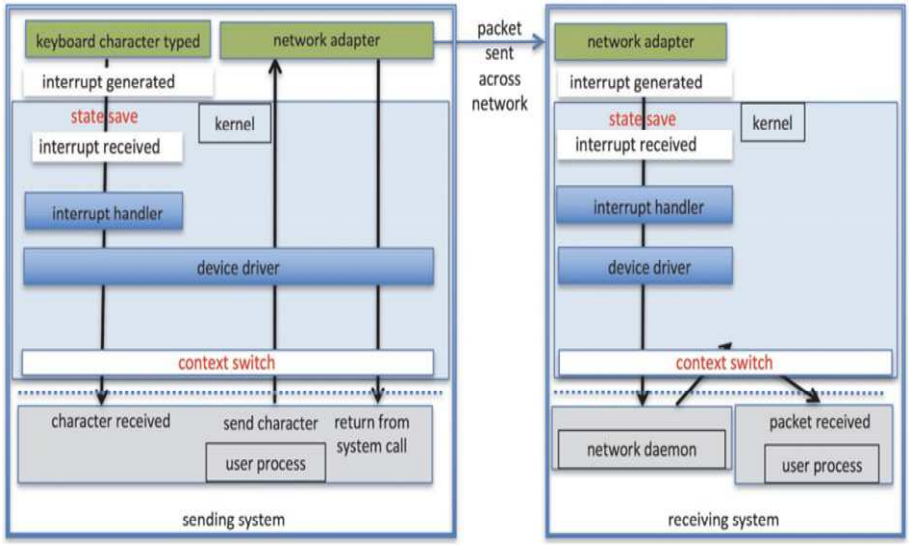




- Several pieces of data are needed to manage open files:
  - File pointer: pointer to last read/write location, per process that has the file open
  - File-open count: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - Disk location of the file: cache of data access information
  - Access rights: per-process access mode information

Introduction to Operating System _ Remote File System

Intercomputer Communication



Intercomputer communications.

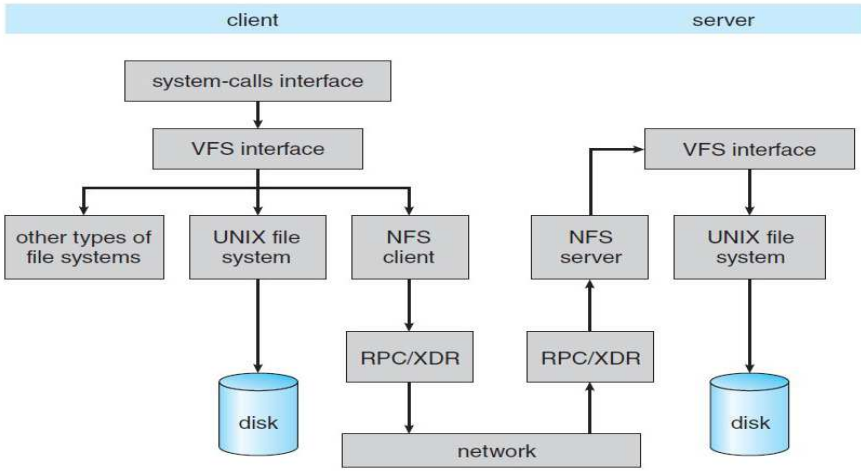


Figure 15.8 Schematic view of the NFS architecture.

Communication with Socket

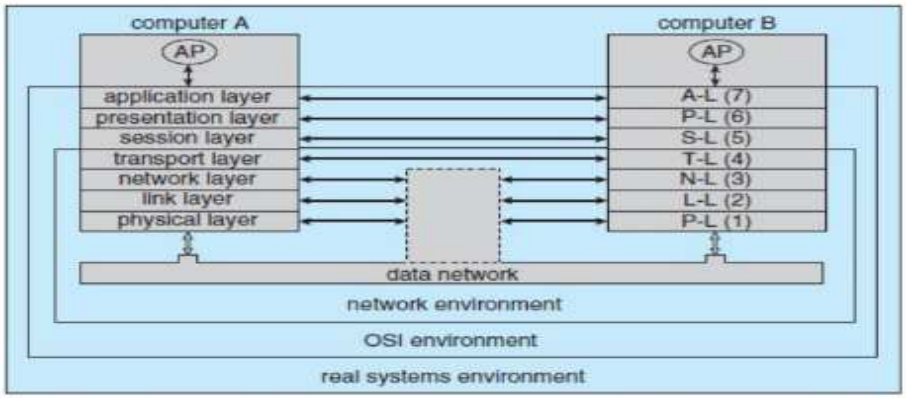


Figure 19.5 Two computers communicating via the OSI network model.

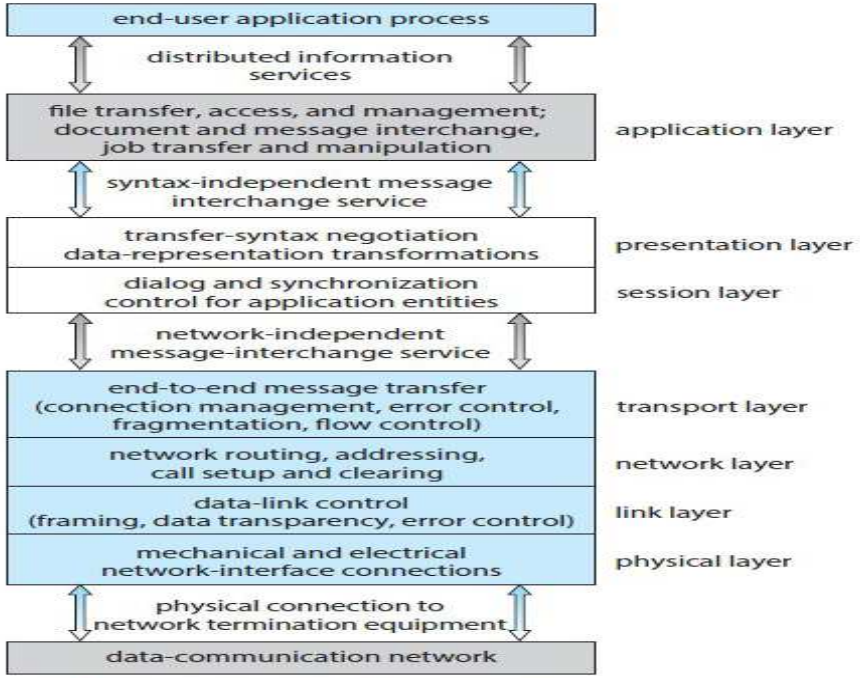


Figure 19.6 The OSI protocol stack.

Introduction to Operating System _ Design and Structure

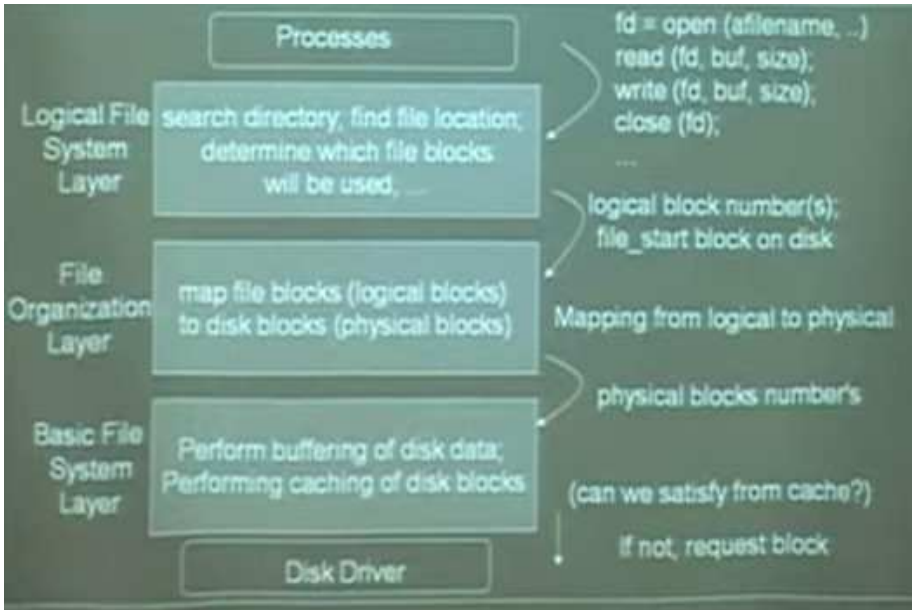
File System Design

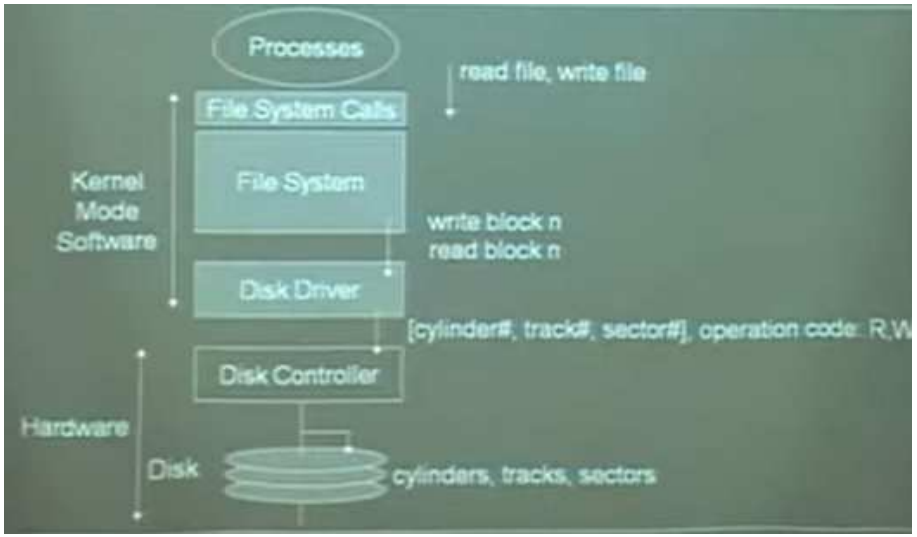
- File System Design Involves
  - 1) Defining File System Interface
    - How file system looks to the user
    - What is a file and its attributes
    - What are the operations
    - (logical) directory structure that can be used to organize files
  - 2) How that file system can be implemented
    - Design algorithms
    - Design data structures (in-memory and on-disk data structures)
    - Map logical file system to physical storage device (disk, tape, etc)
      - Storage device dependent

File Stucture

- File structure
  - Logical storage unit
  - Collection of related information
- File system organized into layers
- File system resides on secondary storage (disks)
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
  - Can also sit on other media (USB disk, CD-ROM, etc). Usually need a different file system
- File control block – storage structure consisting of information about a file
- Device driver controls the physical device

Layered File System





Introduction to Operating System _ File Allocation

Directory Implementation

- **Linear list** of file names with pointer to the data blocks.
  - simple to program
  - time-consuming to execute

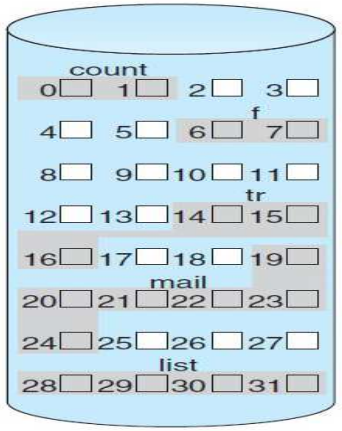
a subdirectory

[abc.txt, address]
[myprog.c, address
[myprog.o, address
...
[data.txt, address]

sitting on disk

- **Hash Table** – linear list with hash data structure.
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

Contiguous Allocation



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Figure 14.4 Contiguous allocation of disk space.

Indexed Allocation

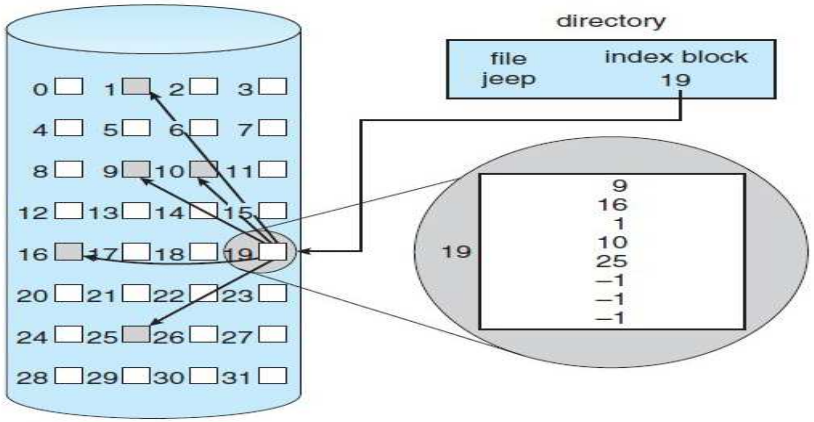


Figure 14.7 Indexed allocation of disk space.

Linked Allocation

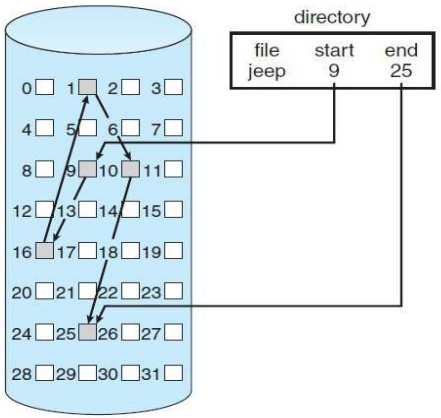


Figure 14.5 Linked allocation of disk space.

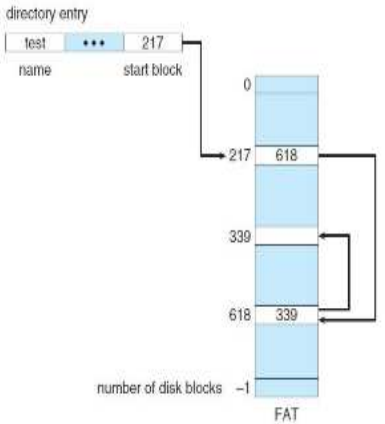
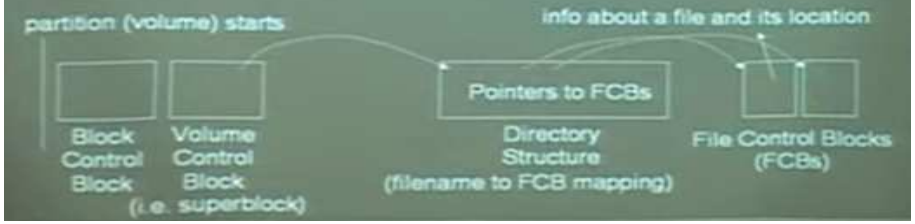


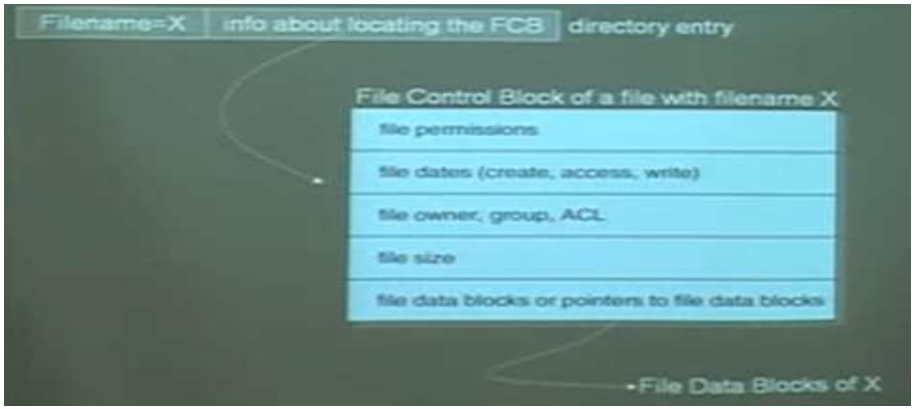
Figure 14.6 File-allocation table.

Introduction to Operating System _ FCB anfd Disk Driver

File Control Block

- Major On-disk Structures (information):
  - Boot control block contains info needed by system to boot OS from that volume
  - Volume control block contains volume details
  - Directory structure organizes the files
  - Per-file File Control Block (FCB) contains many details about the file





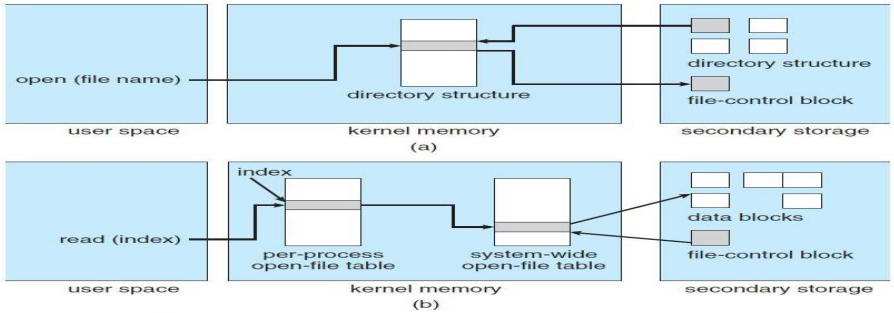
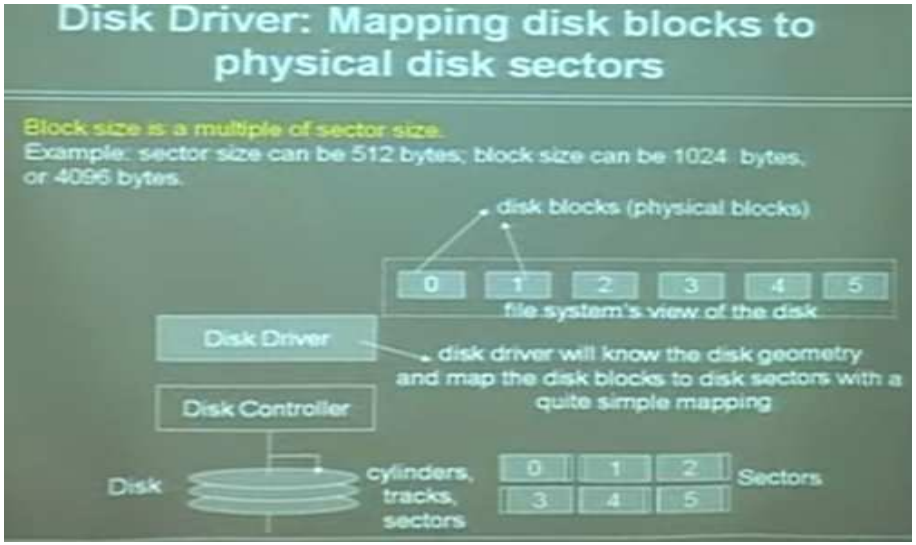
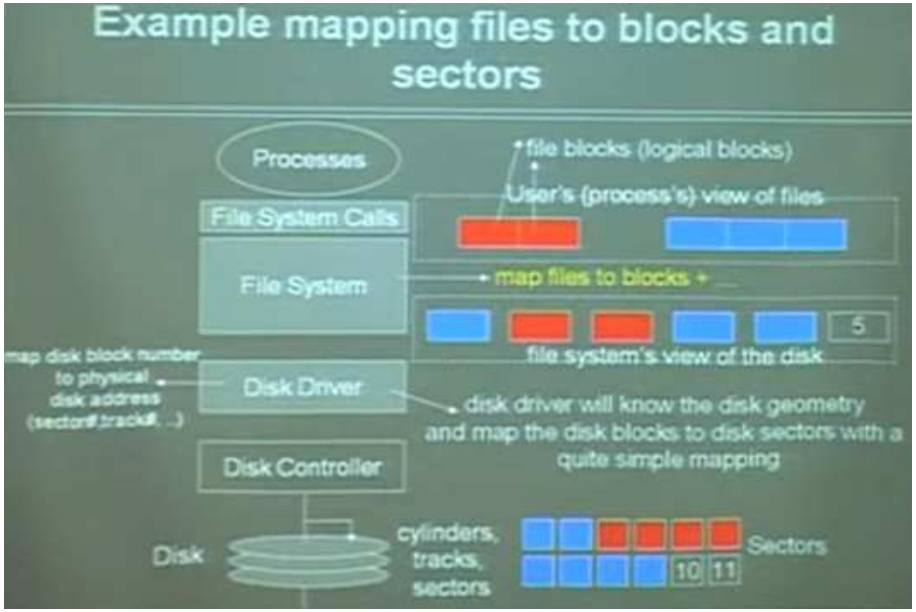


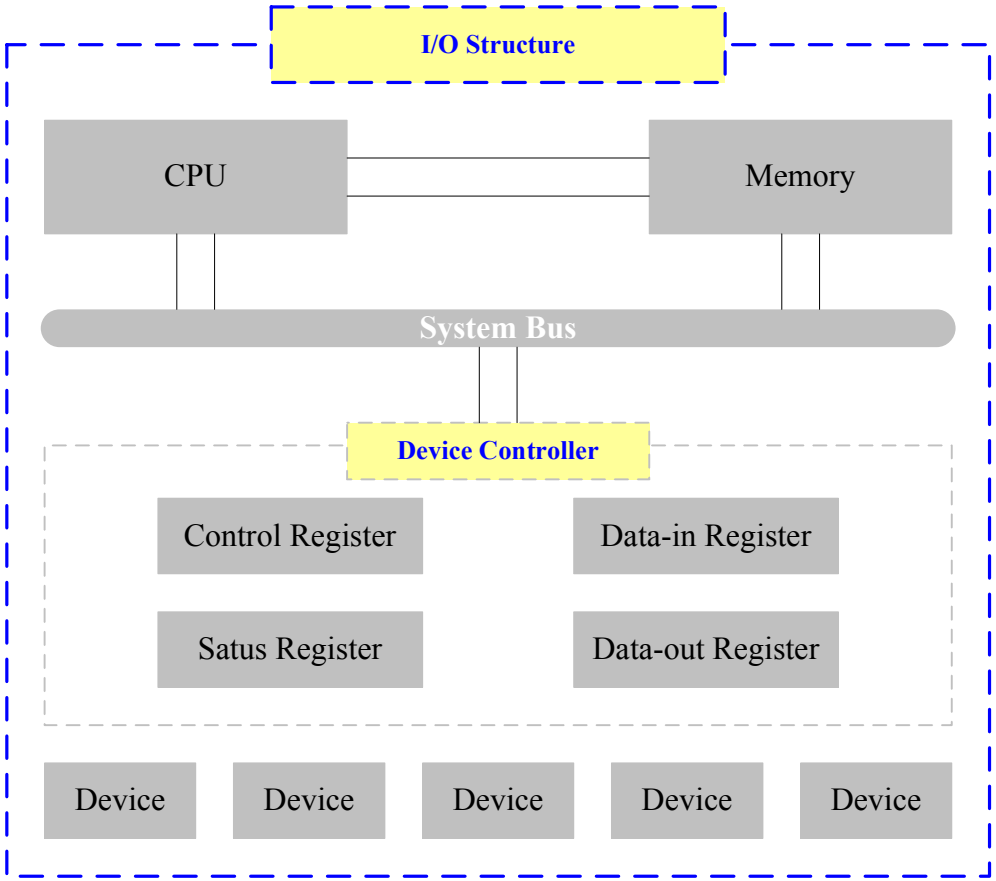
Figure 14.3 In-memory file-system structures. (a) File open. (b) File read .

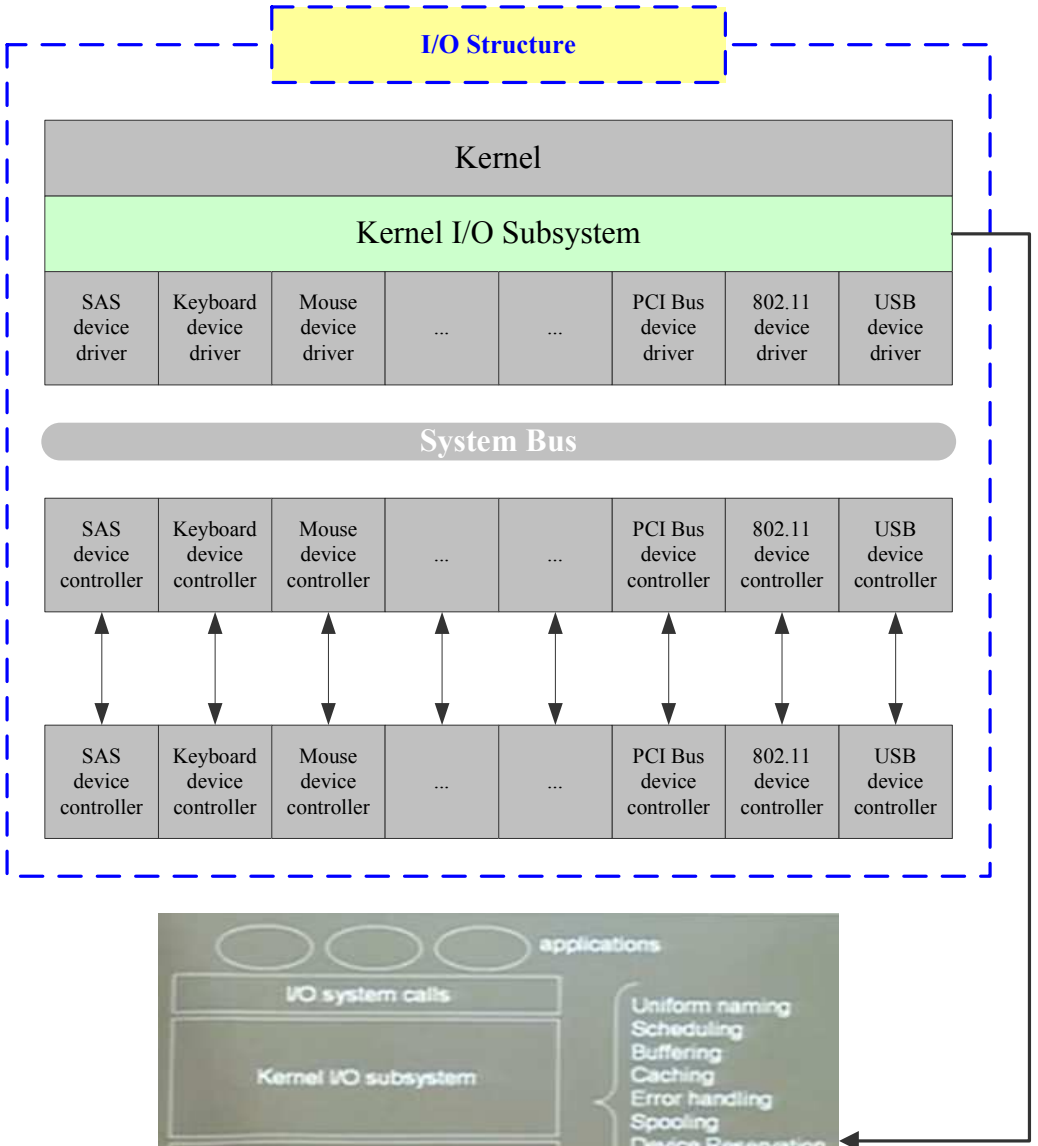
Disk Driver





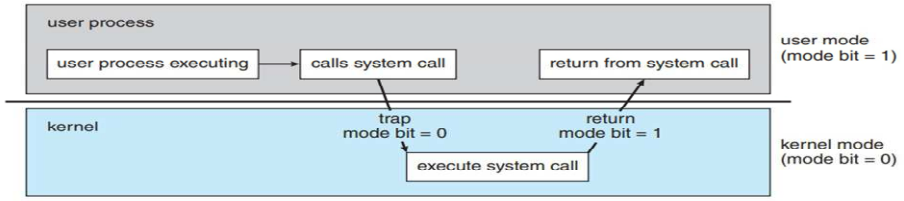
Introduction to Operating System _ I/O Structure



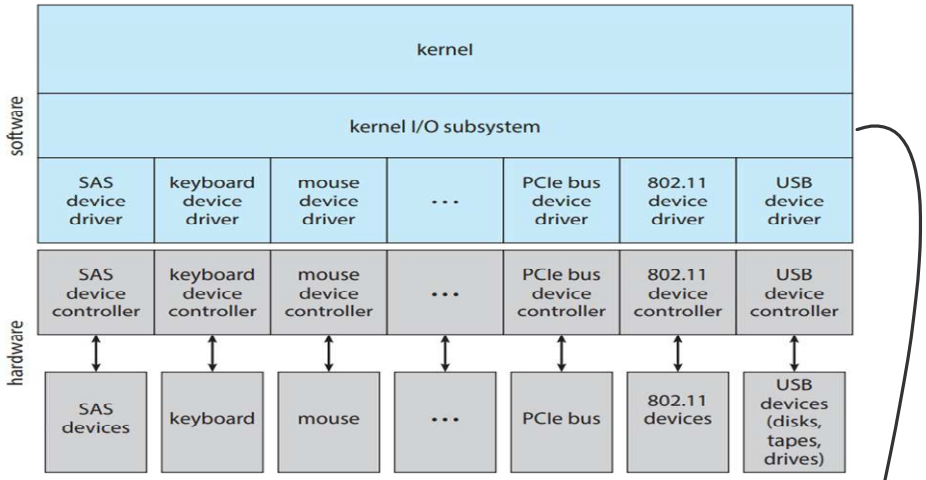


Introduction to Operating System _ A Kernel I/O Structure

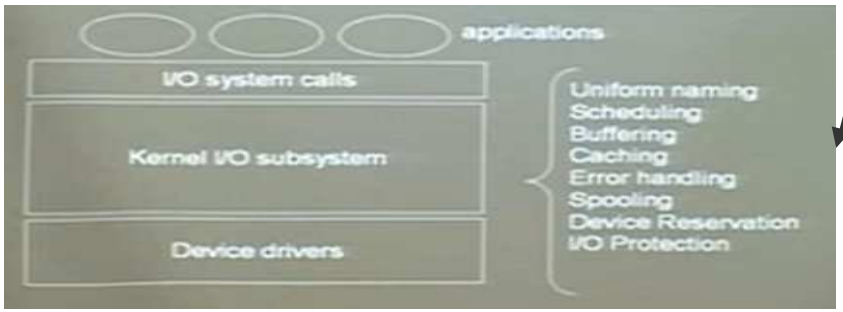
A Kernel I/O Structure



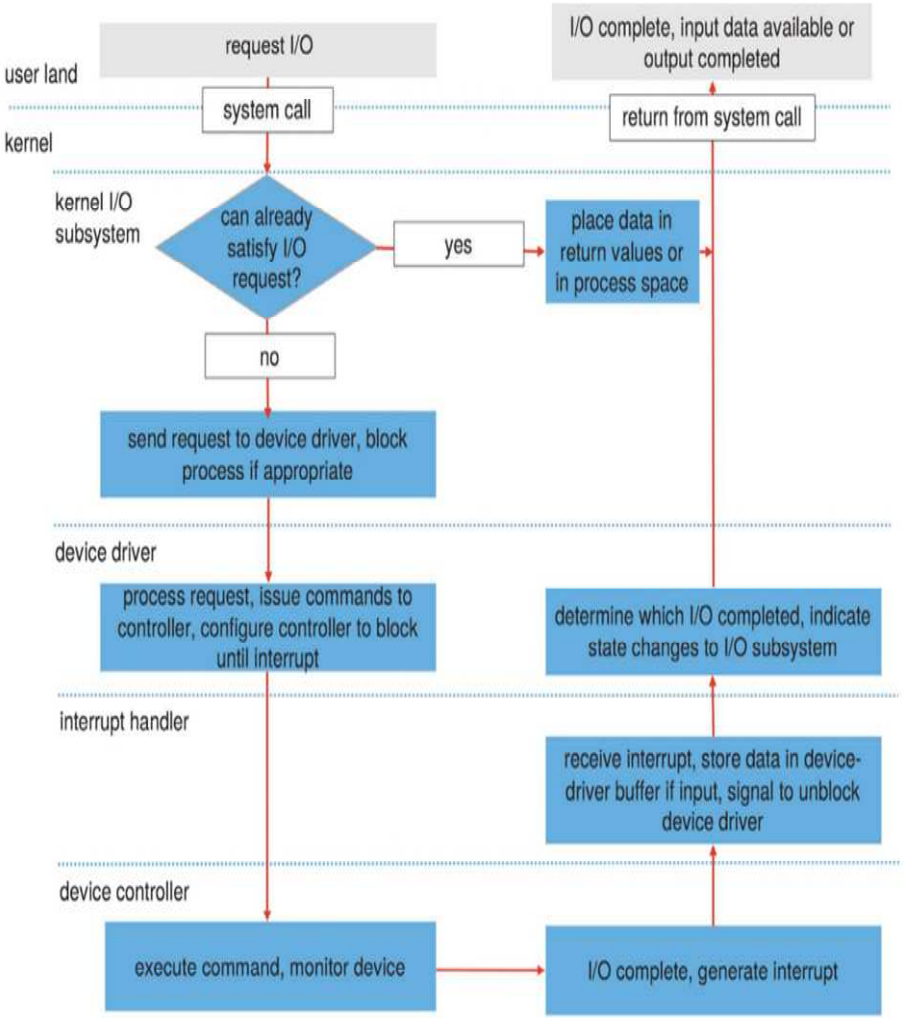
Transition from user to kernel mode.



A kernel I/O structure.



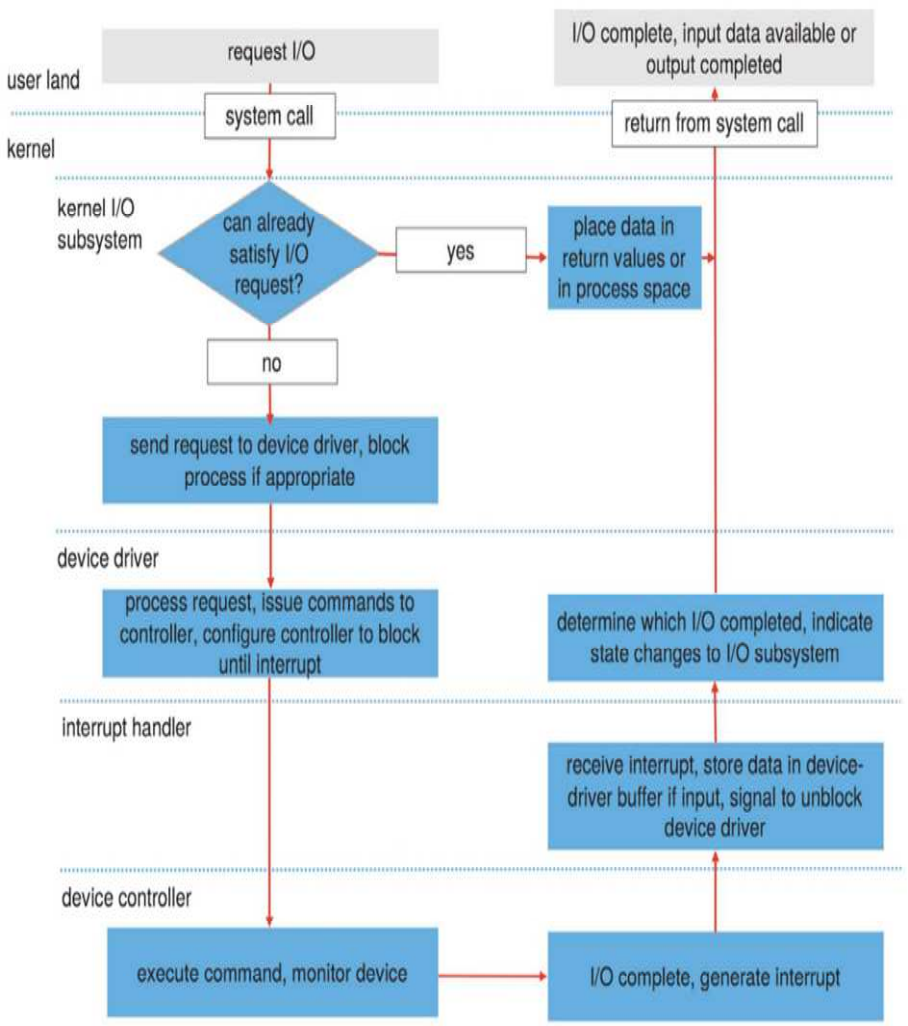
Life Cycle of I/O Request



The life cycle of an I/O request.

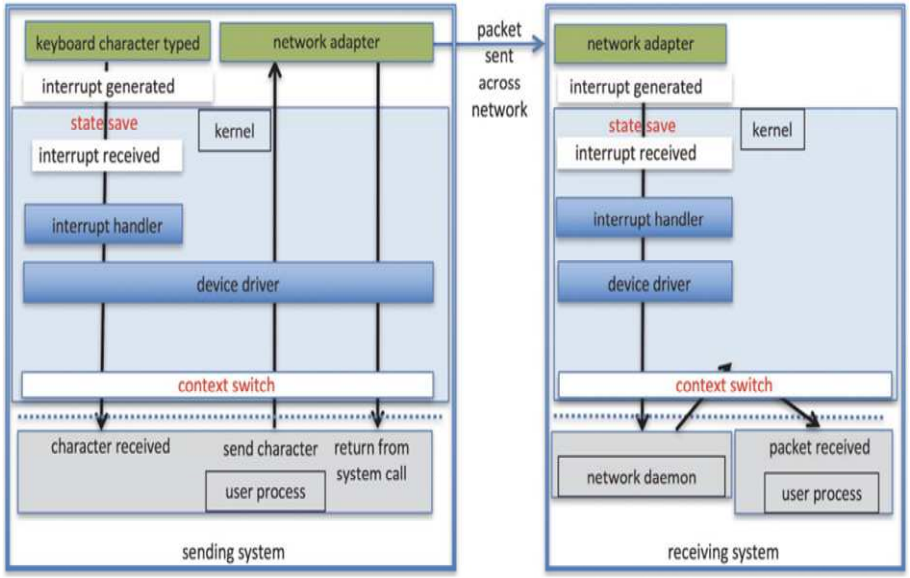
Introduction to Operating System _ I/O Request

Life Cycle of I/O Structure

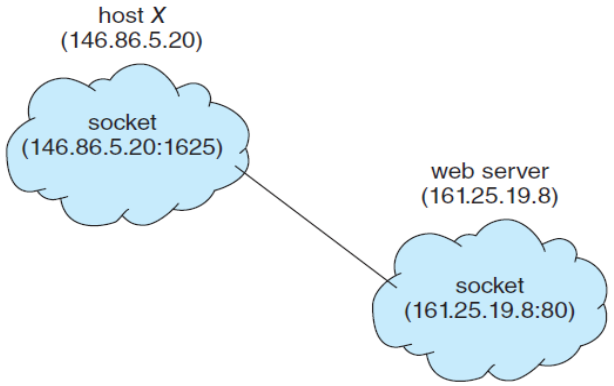


The life cycle of an I/O request.

Intercomputer Communication



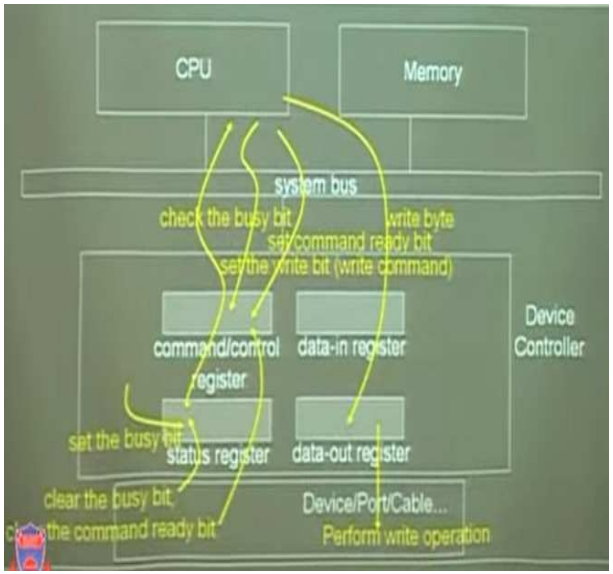
Intercomputer communications.

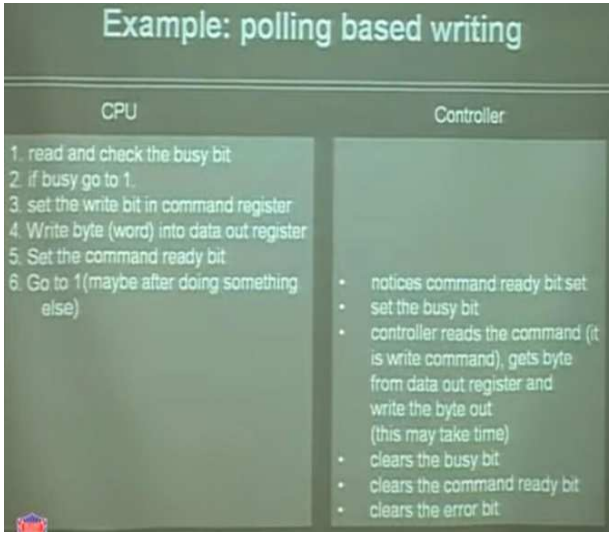


Communication using sockets.

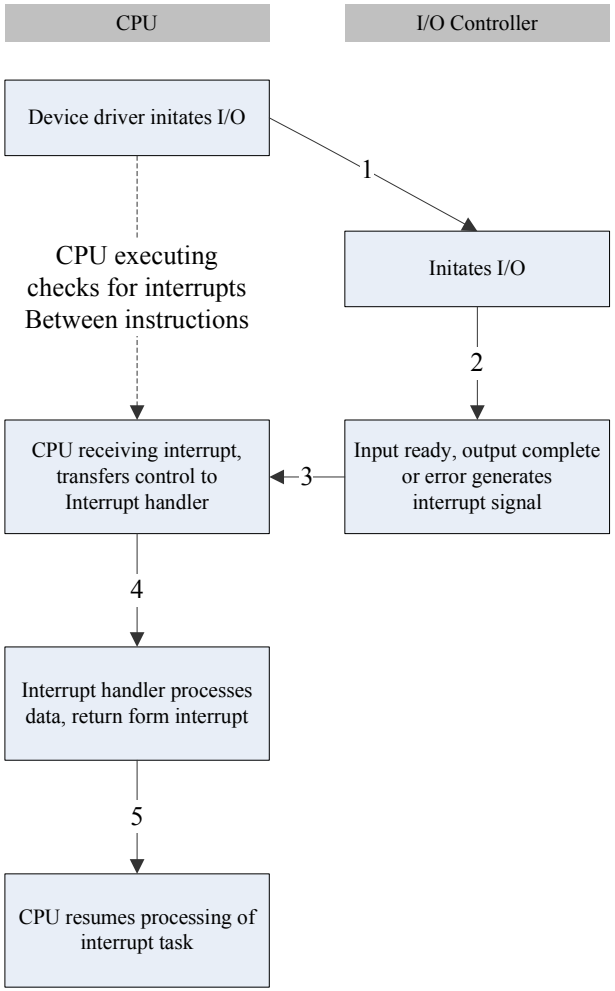
Introduction to Operating System _ I/O System _ Interacting with Device Controller

Polling



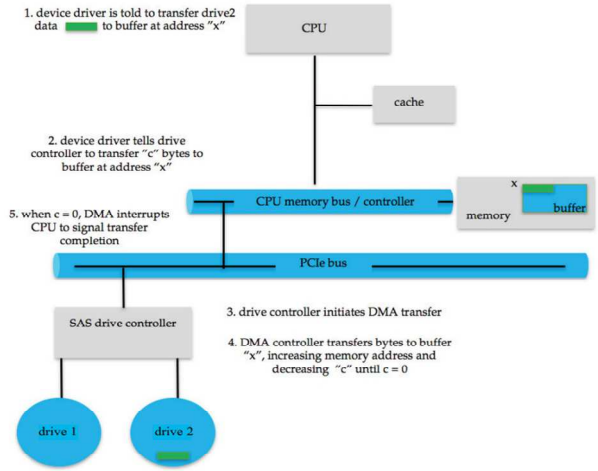


Interrupt Driven

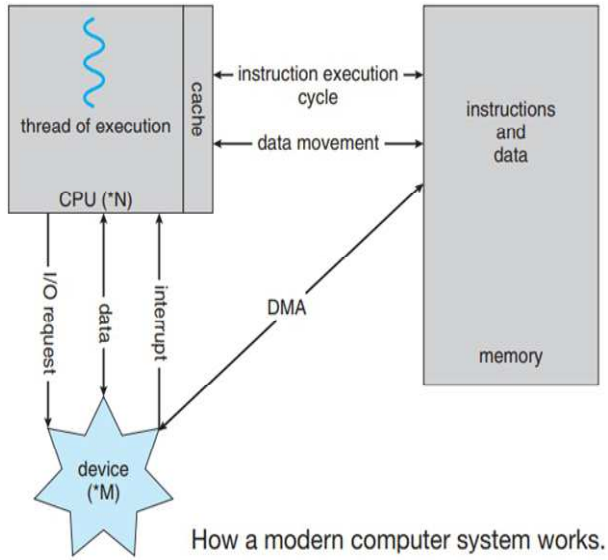


Interrupt-driven I/O Cycle

Interrupt Driven with DMA



Steps in a DMA transfer.



How a modern computer system works.