Operating Systems Principles

cosc1112/cosc1114
School of Science
Semester 2, 2017

Lecture 12 – Course Review

Dr. Ke Deng

ke.deng@rmit.edu.au

RMIT UNIVERSITY

Exam Mark Distribution

2 hours, cover all materials in slides, quizzes, tutorials, and labs.

- Introduction (7 marks)
- Process (9 marks)
- Threads (10 marks)
- CPU scheduling (14 marks)
- Process synchronization (10 marks)
- Dead locks (10 marks)
- Memory management (9 marks)
- File system (12 marks)
- Mass storage and I/O system (13 marks)
- Protection (6 marks)

Exam Question Types

- Multiple Selection (like in quiz)
- Short answer (like in tutorial)
- Read Code (like in tutorial)
- Calculation (like in tutorial)

Consultation Time in Week 13

Location 14-9-12

Wednesday	3:00-4:00
Thursday	3:00-4:00
Friday	3:00-4:00

Ke.deng@rmit.edu.au

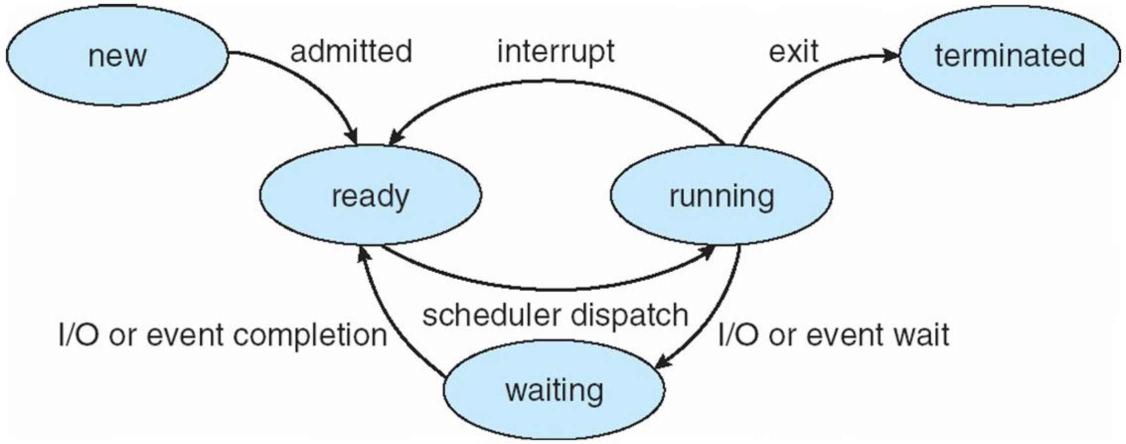
Introduction

- Computer System Structure
- Operating System Definition
- Operating-System Operations
  - **User mode** and **kernel mode**
- Operating System Services
  - services helpful to the user
  - functions ensuring the efficient operation of the system itself
- System Calls
- System Programs

  File manipulation ; Status information sometimes stored in a File modification; Programming language support; Program loading and execution; Communications; Background services; Application programs

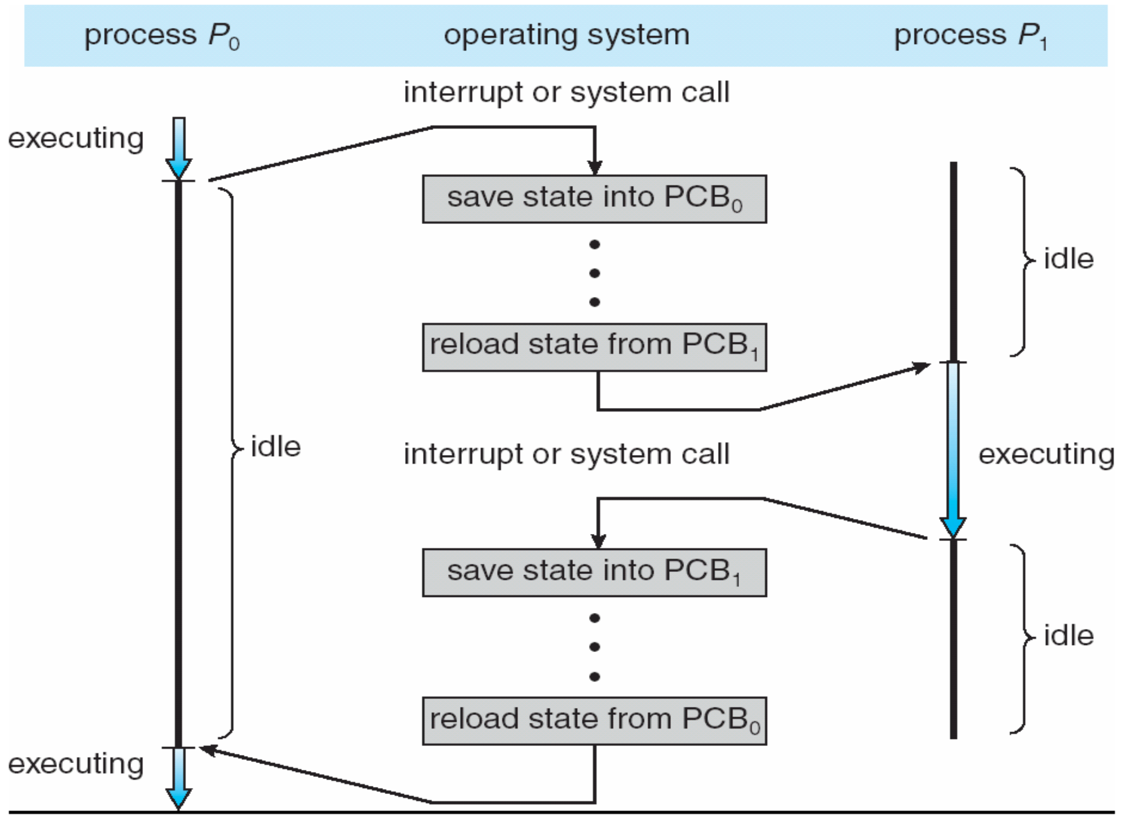
Process

- Process – a program in execution; process execution must progress in sequential fashion
- As a process executes, it changes state
  - **new**: The process is being created; **running**: Instructions are being executed; **waiting**: The process is waiting for some event to occur; **ready**: The process is waiting to be assigned to a processor; **terminated**: The process has finished execution



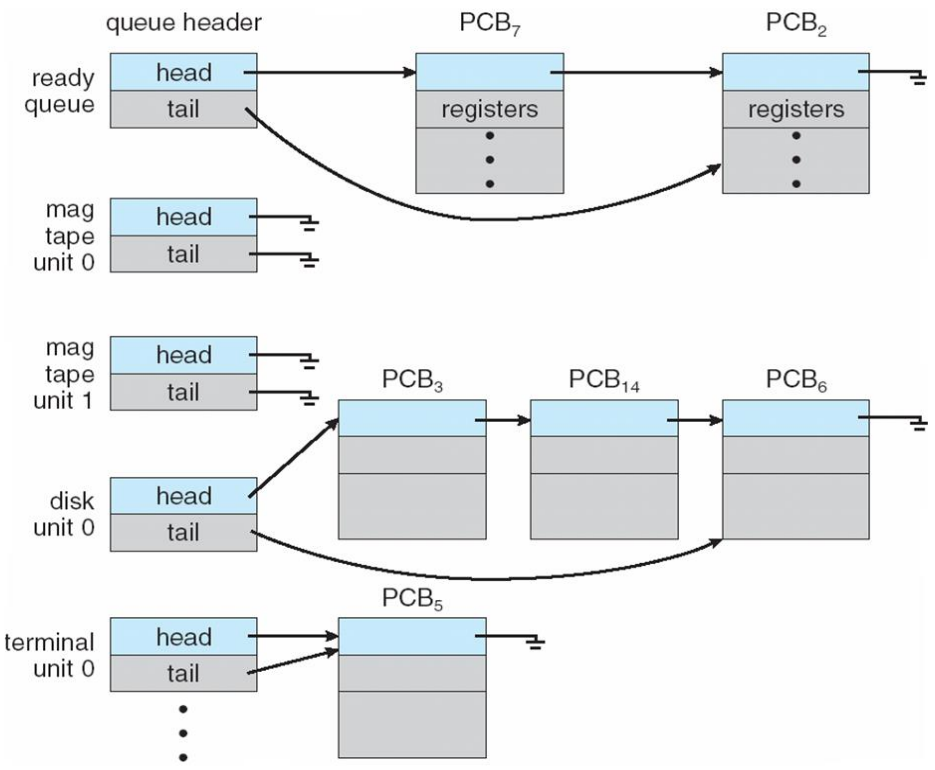
Process (cont.)

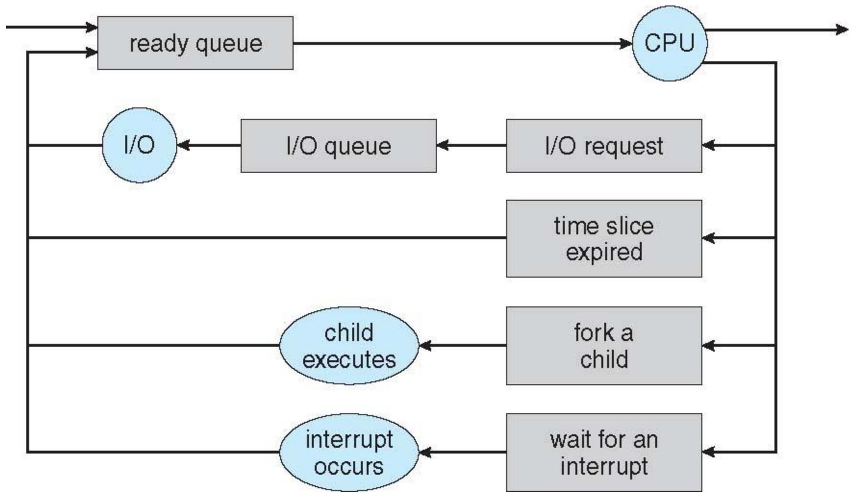
- Maximize CPU use, quickly switch processes onto CPU for time sharing



Process (cont.)

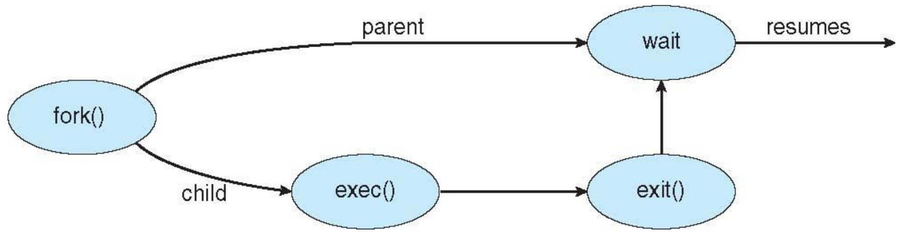
- Process scheduler selects among available processes for next execution on CPU; Maintains scheduling queues of processes
  - Job queue – set of all processes in the system
  - Ready queue – set of all processes residing in main memory, ready and waiting to execute
  - Device queues – set of processes waiting for an I/O device
  - Processes migrate among the various queues





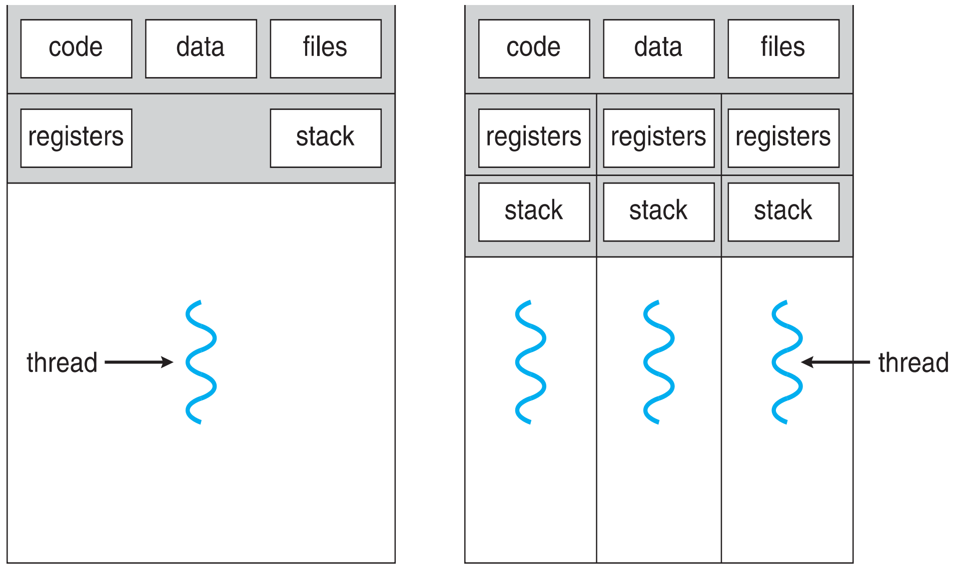
Process (cont.)

- Types of scheduler
  - Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue
  - Medium-term scheduler can be added if degree of multiple programming needs to decrease - Remove process from memory, store on disk, bring back in from disk to continue execution: swapping
- Process creation - UNIX examples
  - fork() system call creates new process
  - exec() system call used after a fork() to replace the process' memory space with a new program



Thread

- Process creation is heavy-weight while thread creation is light-weight, can simplify code, increase efficiency



single-threaded process multithreaded process

- Amdahl's Law: Identifies performance gains from adding additional cores to an application that has both serial and parallel components

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

S is serial portion
N processing cores

Thread (cont.)

- User threads - management done by user-level threads library
- Kernel threads - Supported by the Kernel
  - one-to-one, many-to-one, many-to-many
- Three primary thread libraries:
  - POSIX Pthreads
  - Windows threads
  - Java threads

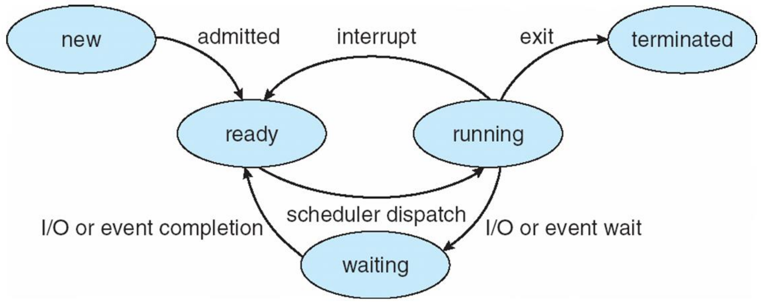
Thread (cont.)

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

flag	meaning
`CLONE_FS`	File-system information is shared.
`CLONE_VM`	The same memory space is shared.
`CLONE_SIGHAND`	Signal handlers are shared.
`CLONE_FILES`	The set of open files is shared.

CPU Scheduling

- Short-term scheduler selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  - Switches from running to waiting state (nonpreemptive)
  - Switches from running to ready state
  - Switches from waiting to ready
  - Terminates (nonpreemptive)



- Scheduling Algorithm Optimization Criteria
  - Max CPU utilization, Max throughput, Min turnaround time, Min waiting time, Min response time

CPU Scheduling (cont.)

- Scheduling Algorithms
  - First- Come, First-Served (FCFS)
  - Shortest-Job-First (SJF) - Shortest-remaining-time-first
  - Priority
  - Round Robin (RR)
- Ready queue is partitioned into separate queues
  - Multilevel Queue, Multilevel Feedback Queue

Process Synchronization

- Race Condition
  - When several processes access and manipulate the same data concurrently, the situation that the outcome of the execution depends on the particular order in which the access takes place is called a
- A process may have critical section segment of code
  - Process may be changing common variables, updating table, writing file, etc.
  - When one process in critical section, no other may be in its critical section

Critical section problem is to design protocol to solve this problem

- Each process must ask permission to enter critical section in entry section, may follow critical section with exit section, then remainder section
- Solutions of Critical section problem
  - Peterson’s solution
  - Hardware based solution
  - Semaphore

Process Synchronization (cont.)

- Deadlock

  Two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes

- Solution to Critical-Section Problem
  - **Mutual Exclusion** - If process $\boldsymbol{P_i}$ is executing in its critical section, then no other processes can be executing in their critical sections
  - **Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely
  - **Bounded Waiting** - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted

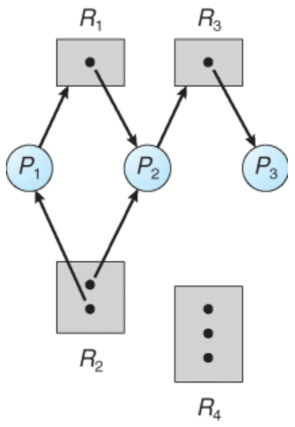
- Classical problems used to test newly-proposed synchronization schemes
  - Bounded-Buffer Problem
  - Readers and Writers Problem

Deadlock

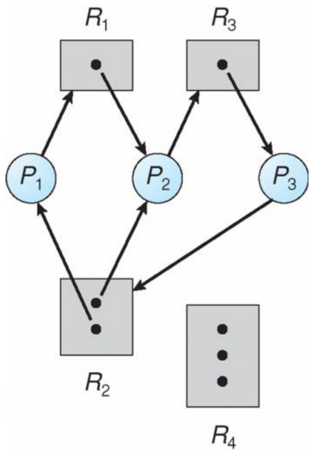
Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion**: only one process at a time can use a resource
- **Hold and wait**: a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption**: a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait**: there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

Resource Allocation Graph



Resource Allocation Graph Without A Deadlock



Resource Allocation Graph With A Deadlock

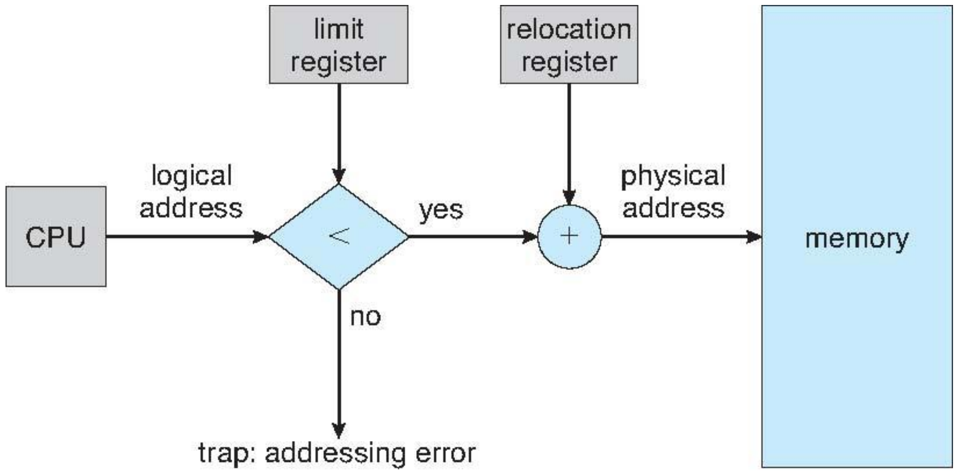
Deadlock

Methods for Handling Deadlocks

- Ensure that the system will ***never*** enter a deadlock state:
  - Deadlock prevention
  - Deadlock avoidence
- Allow the system to enter a deadlock state and then recover
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

Memory Management

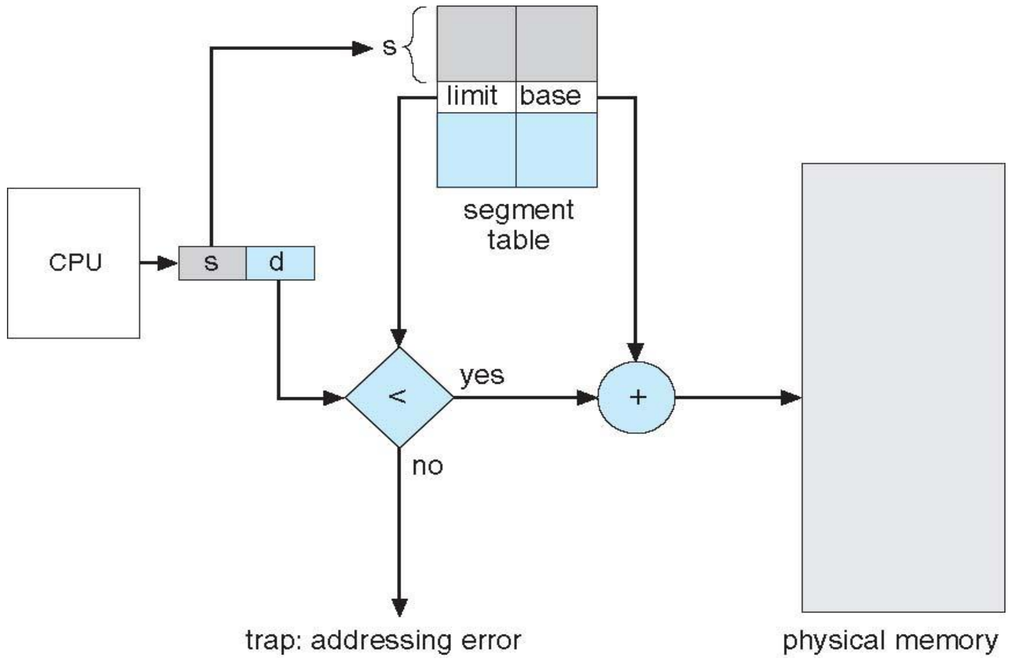
- Swapping - A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Backing store, Roll out roll in, swap time,
- Contiguous Memory Allocation
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - First-fit, worst fit, best fit
  - External Fragmentation



Memory Management

- Segmentation
  - Logical address consists of two tuple

    <segment-number, offset>



Memory Management (cont.)

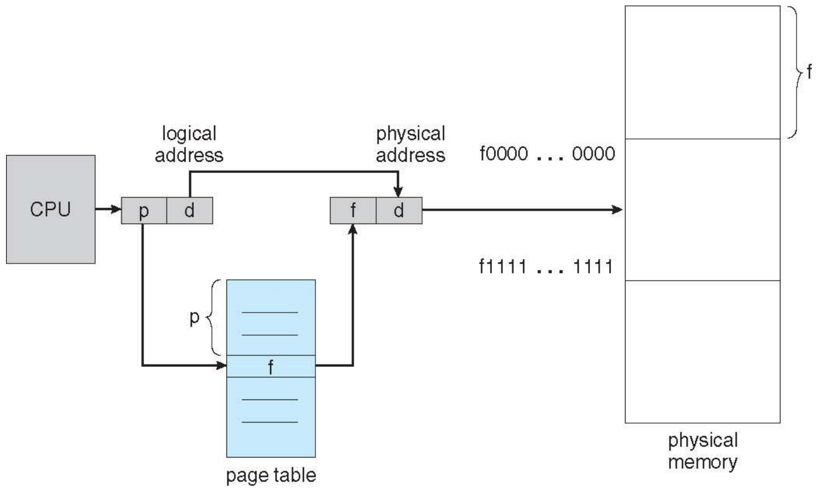
- Paging
  - Address generated by CPU is divided into:

    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
  - Internal Fragmentation
  - Page Table

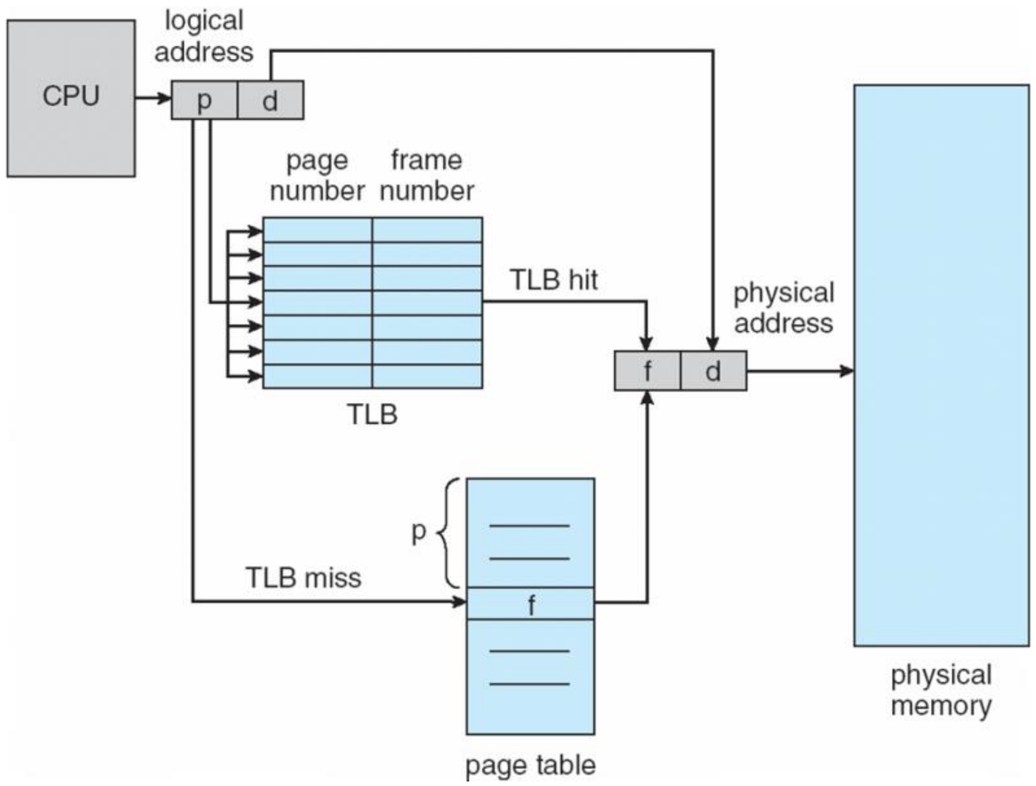
    Page-table base register (PTBR) points to the page table

    Page-table length register (PTLR) indicates size of the page table



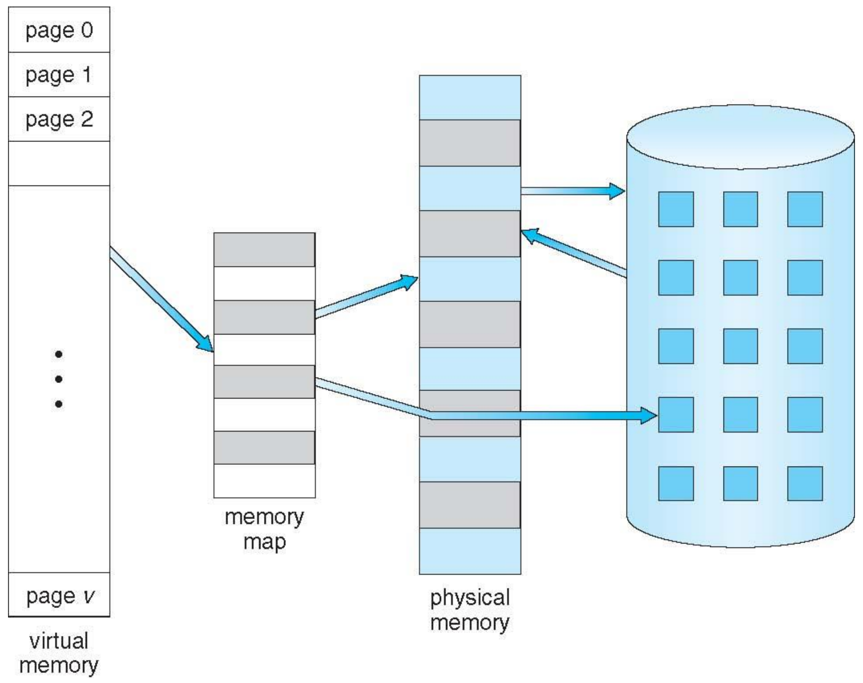
Memory Management (cont.)

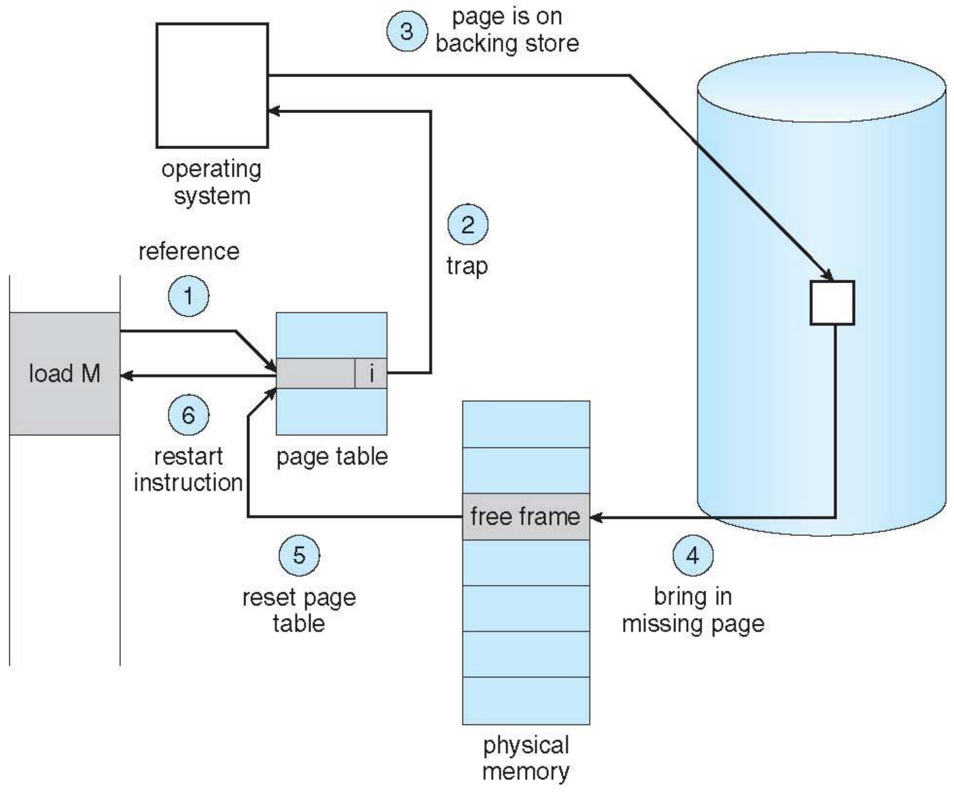
- Associative memory or translation look-aside buffers (TLBs) Hierarchical Page Tables



Memory Management (cont.)

- Virtual Memory
  - Code needs to be in memory to execute, but entire program rarely used
  - Swapper that deals with pages is a pager
  - Page fault



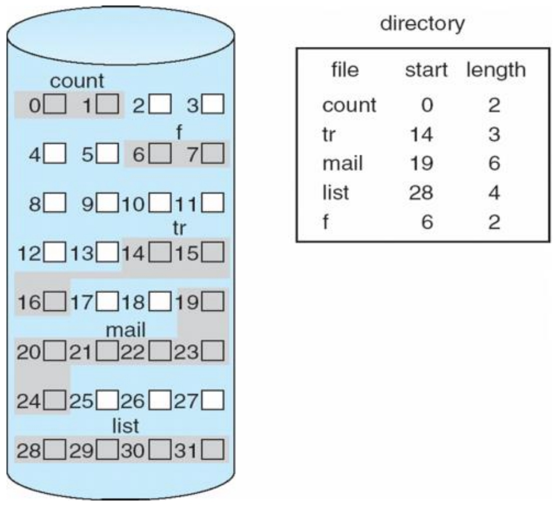


File System

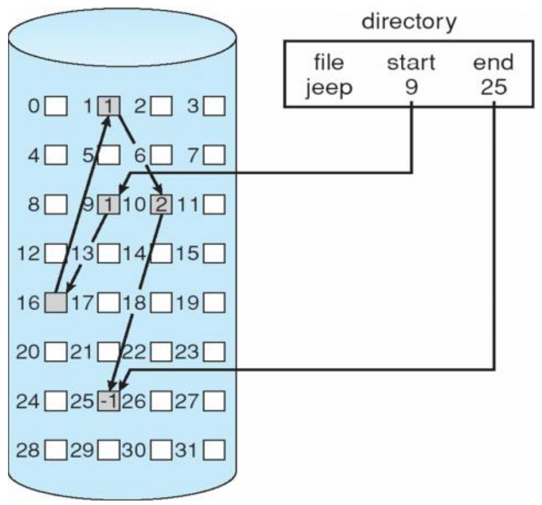
- Access Methods
  - Sequential Access on Direct-access File
- Disk and Directory Structure
  - Partitions, volume,
  - Systems frequently have many file systems, some general- and some special-purpose
  - Single-Level Directory, Two-Level Directory, Tree-Structured Directories, Acyclic-Graph Directories (guarantee no cycles), General Graph Directory

File System (cont.)

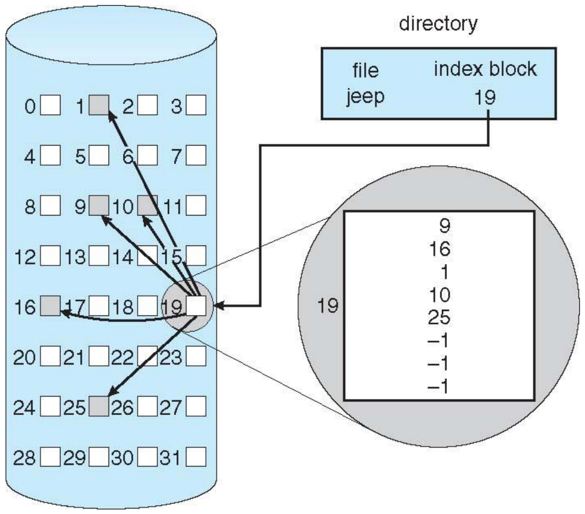
- Allocation Methods
  - Contiguous allocation
  - Linked allocation
  - Indexed Allocation



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



file	start	end
jeep	9	25



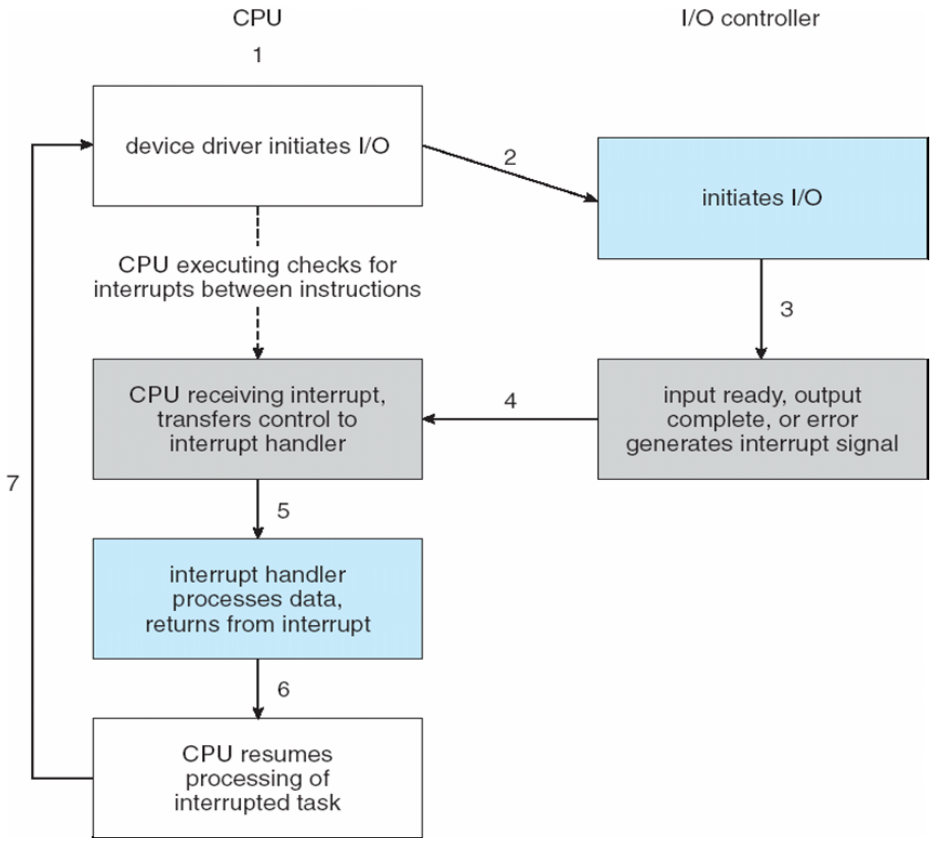
file	index block
jeep	19

Mass storage & I/O System

- Mass storage
  - Magnetic disks, Access Latency
  - Solid-State Disks
  - Host-Attached Storage
  - Network-Attached Storage
- Disk Scheduling
  - FCFS
  - SSTF (Shortest Seek Time First)
  - SCAN, Look
  - C-SCAN, C-look
- RAID Structure (Redundant Array of Inexpensive Disks)
  - RAID 0, RAID 1, RAID 5, RAID 10

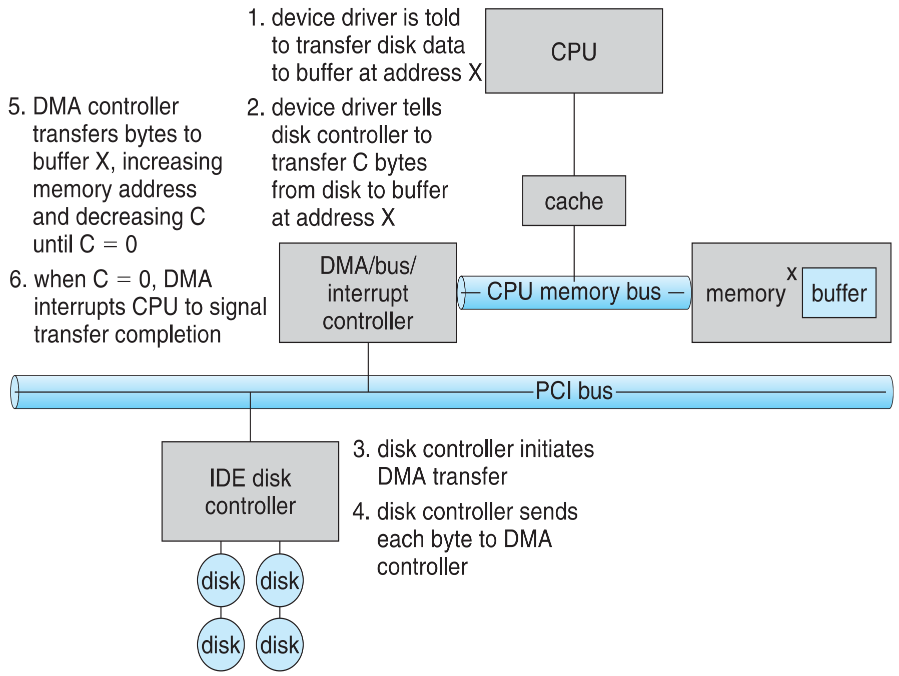
Mass storage & I/O System (cont.)

- I/O instructions control devices
  - Devices usually have registers where device driver places commands, addresses, and data to write, or read data from registers after command execution
  - For each byte of I/O
    1. Read busy bit from status register until 0 (busy-wait cycle)
    1. Host sets read or write bit and if write copies data into data-out register
    1. Host sets command-ready bit
    2. Controller sets busy bit, executes transfer
    3. Controller clears busy bit, error bit, command-ready bit when transfer done
  - Interrupt-Driven I/O Cycle

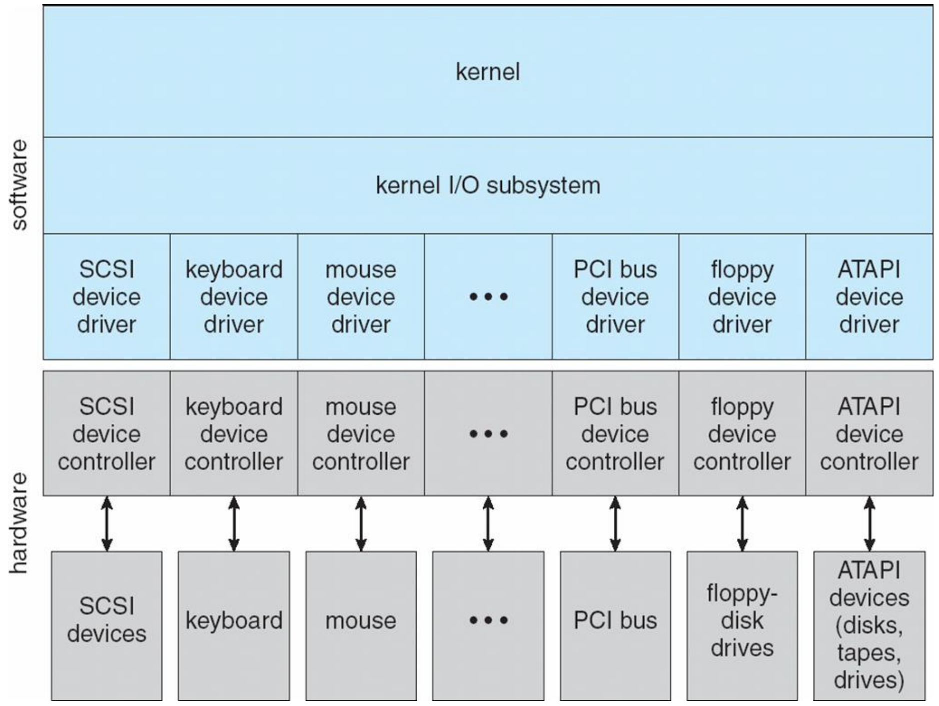

CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task
7

Mass storage & I/O System (cont.)

- Direct Memory Access is used to avoid **programmed I/O** (one byte at a time) for large data movement
- Application I/O Interface
  - I/O system calls encapsulate device behaviors in generic classes
  - Device-driver layer hides differences among I/O controllers from kernel



1. device driver is told to transfer disk data to buffer at address X
CPU
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
cache
6. when C = 0, DMA interrupts CPU to signal transfer completion
DMA/bus/interrupt controller
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
disk
disk
disk
disk



Protection

- Protection
  - Principle of least privilege
    - Access-right = <object-name, rights-set>; Domain = set of access-right
  - Access Matrix
    - owner of O_i
    - copy op from O_i to O_j (denoted by "*")
    - control – D_i can modify D_j access rights
    - transfer – switch from domain D_i to D_j
  - Implementation of Access Matrix
    - Global table
    - Access lists for objects
    - Capability list for domains
    - Lock-key

End of Course
Semester 2, 2017