TDIU11
Operating systems

Introduction
Interrupt, System Calls
Operating System Operations

[SGG7/8/9]
Chapter 1.1-1.7
Chapter 2.3-2.5

Christoph Kessler, IDA,
Linköpings universitet.

What is an Operating System (OS)?

- A program that acts as an **intermediary** between a **user** of a computer and the computer **hardware**.
- Operating system goals:
  - Execute user programs in a well-defined environment.
  - Make the computer system convenient to use.
  - Administrate system resources.
  - Enable efficient use of the computer hardware.
- An operating system provides an **environment** within which other programs can do useful work; the OS does not perform any "useful" function itself.

What Operating Systems Do

- Depends on the point of view
- Users want convenience, **ease of use** and **good performance**
  - Don't care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

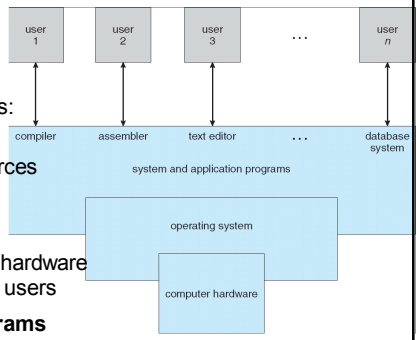
Operating System Definition (1)

- OS is a **resource allocator**
  - Manages all resources of a computer system
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

Operating System Definition (2)

- No universally accepted definition
  - "Everything a vendor ships when you order an operating system" is a good approximation
  - But varies wildly
- "The one program running at all times on the computer" is called the **kernel.**
  - Everything else is either a **system program** (ships with the operating system) or an **application program.**

Computer System Structure

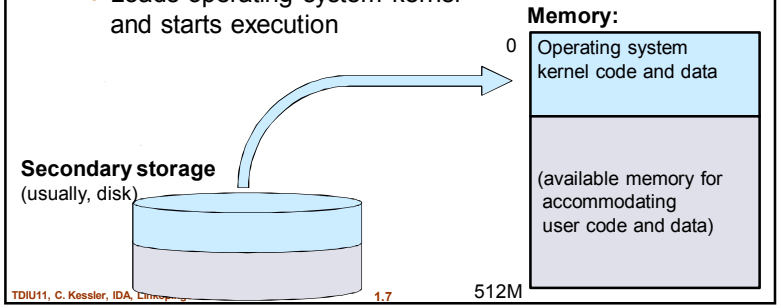


Computer system
can be divided into 4 main layers:

- **Hardware**
  provides basic computing resources
  - CPU, memory, I/O devices
- **Operating system**
  controls and coordinates use of hardware among various applications and users
- **System and Application programs**
  define the ways in which the system resources are used to solve the computing problems of the users
  - Word processors, compilers, web browsers, database systems, games
- **Users**
  - People, machines, other computers

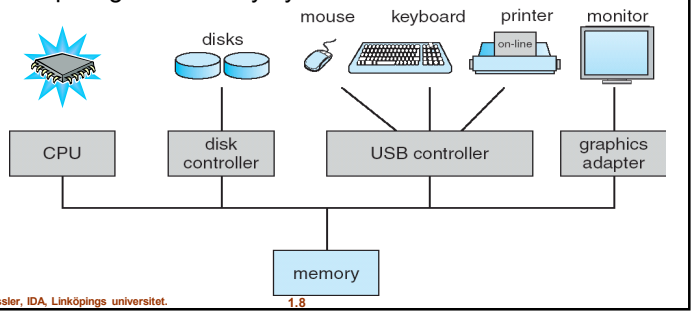
Computer Startup

- **Bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of the system
  - Loads operating system kernel and starts execution



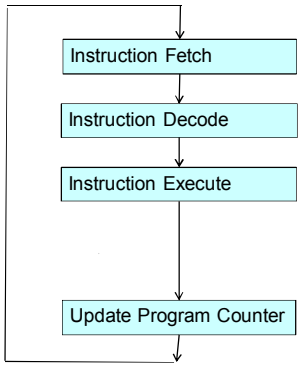
Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connected through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices, competing for memory cycles



Background: Interrupt

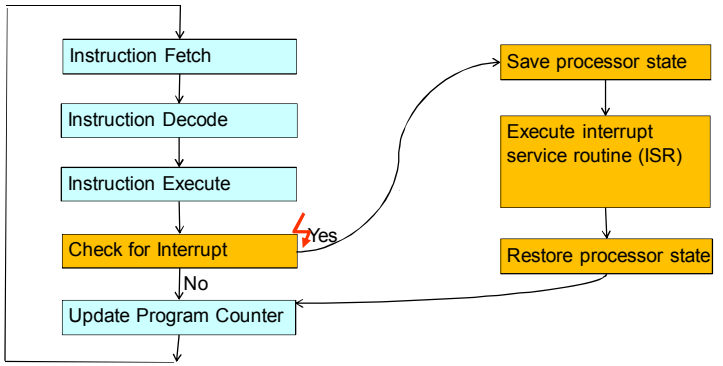
- Program execution (von-Neumann cycle) by a processor



No way to react to events not explicitly anticipated in the (user) program code

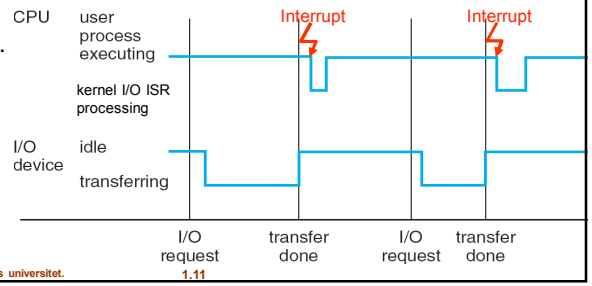
Background: Interrupt

- Program execution (von-Neumann cycle) by a processor with interrupt logic



CPU – I/O Device Interaction (1)

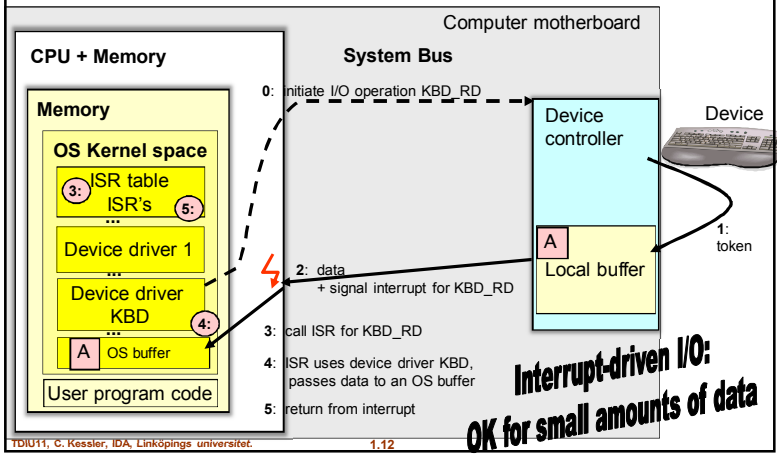
- I/O devices and the CPU can execute concurrently.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an *interrupt*.



Remark: Alternative to interrupt usage: *Polling / Busy-waiting*, see [SGG7] Ch. 13.2.1

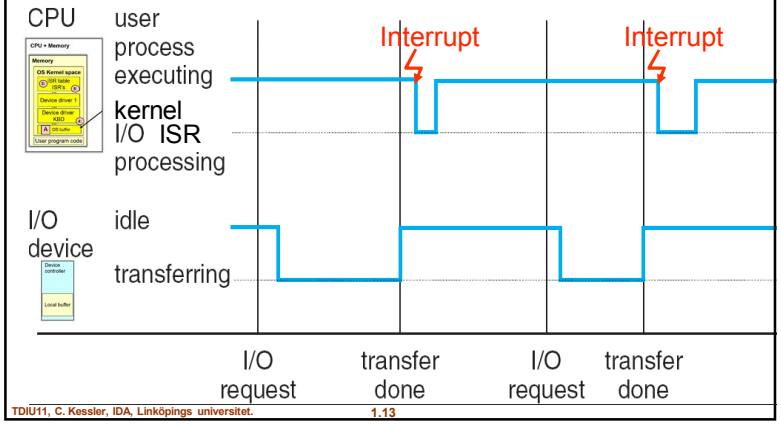
I/O Interaction using Interrupt

- **Example**: Read from the keyboard (KBD)



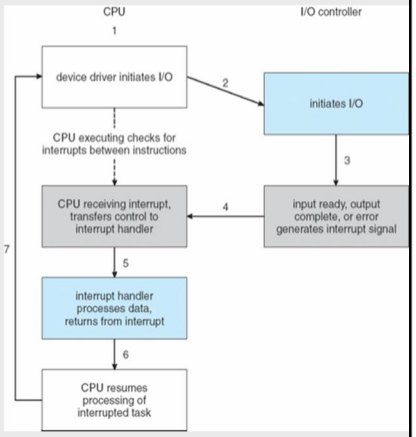
Interrupt Timeline

for a device sending input



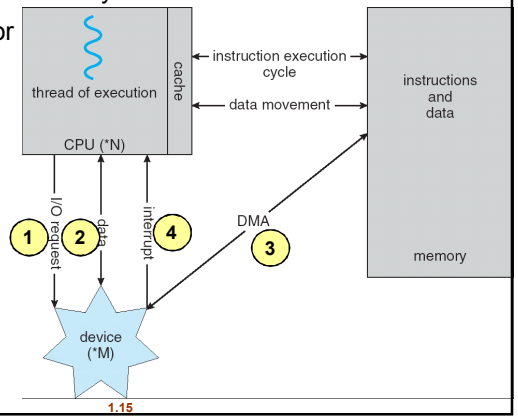
I/O Interaction using Interrupt

- The device driver loads the appropriate registers within the device controller
- The device controller determines what action to take based on the registers
  - "read a character from the keyboard"
- The controller starts the transfer of data from the device to its local buffer
- Once the transfer is complete, the device controller informs the device driver via an interrupt that it has completed the transfer.



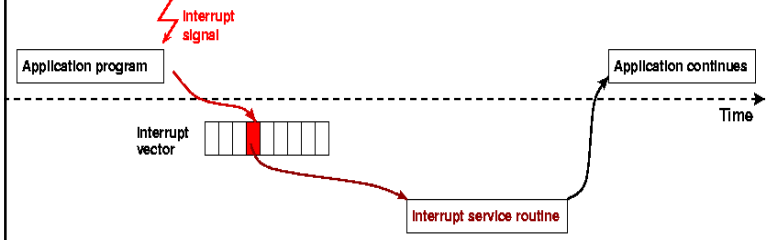
CPU – I/O Device Interaction (2)

- **DMA = Direct Memory Access**
  - allows for parallel activity of CPU and I/O data transfer
  - Moe efficient for large volume data transfer



Interrupt (1)

- Interrupt transfers control to an ***interrupt service routine***, generally through the ***interrupt vector (IRV)***, a branch table that contains the start addresses of all the service routines.



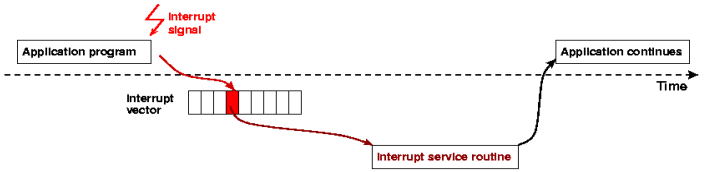
- Interrupt architecture must save the address of the interrupted instruction.
- How to determine which type of interrupt has occurred?
  - ***polling***
  - ***vectored* interrupt system**: interrupt number indexes IRV

Interrupt (2)

- A ***trap*** is a *software*-generated interrupt caused either by an error or a user request.
  - Examples: Division by zero;
    Request for OS service
- An operating system is ***interrupt* driven**.

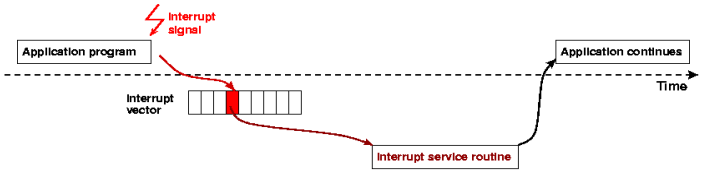
Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system
- Separate segments of code (**Interrupt service routines**) determine what action should be taken for each type of interrupt



Interrupt Handling

- Hardware + OS preserves the state of the CPU by
  - Storing registers and program counter (address of interrupted instruction)
- Determines which type of interrupt has occurred:
  - **Polling** (Continually checking a non-busy bit in device controllers' status register)
  - **Vectored interrupt system**: Interrupt signal, number indexes into IRV table
- Separate segments of code determine what action should be taken for each type of interrupt

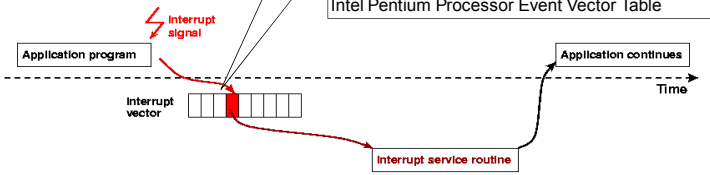


Interrupt Handling

- Hardware + OS preserves the s
  - Storing registers and program co
- Determines which type of interr
  - **Polling** (Continually checking a n
  - **Vectored interrupt system**: Int
- Separate segments of code
  for each type of interrupt

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Intel Pentium Processor Event Vector Table



TDIU11
Operating systems

System Calls

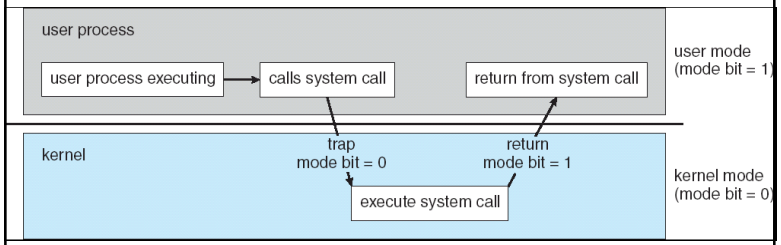
Introduction

More in TDIU16: System call API,
Passing parameters, Types of system calls

Christoph Kessler, IDA,
Linköpings universitet.

Dual mode, System calls

- **Dual-mode** operation
  allows OS to protect itself and other system components
  - ***User mode*** and ***kernel mode*** (*supervisor mode, privileged mode*)
    - ***Privileged instructions*** only executable in kernel mode
    - ***System call*** changes mode to kernel, on return resets it to user
  - ***Mode bit*** provided by hardware

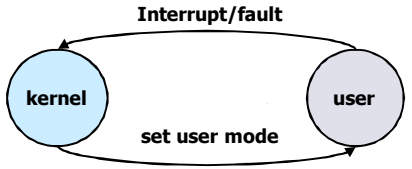


Dual-Mode Operation

- Sharing system resources requires the operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Hardware support (**mode bit**) to differentiate between at least two modes of operations.
- **User mode**
  - Execution done on behalf of a user
  - Access only to memory addresses owned by the process
- **Kernel mode** (also *supervisor mode* or *system mode*)
  - Execution done on behalf of operating system.
  - **Privileged instructions** are executable
    (= instructions that may be harmful,
    e.g., system login, set priorities, system halt, etc.)
  - Unrestricted memory access

Dual-Mode Operation (Cont.)

- When an interrupt or fault occurs, hardware switches to kernel mode.
- System calls – call OS service

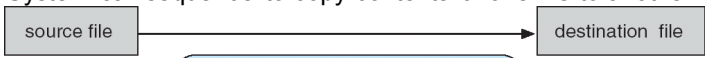


Remark:

Increasingly, CPUs support **multi-mode** operations for virtualization, i.e. virtual machine manager (VMM) mode for guest VMs

Example of System Calls

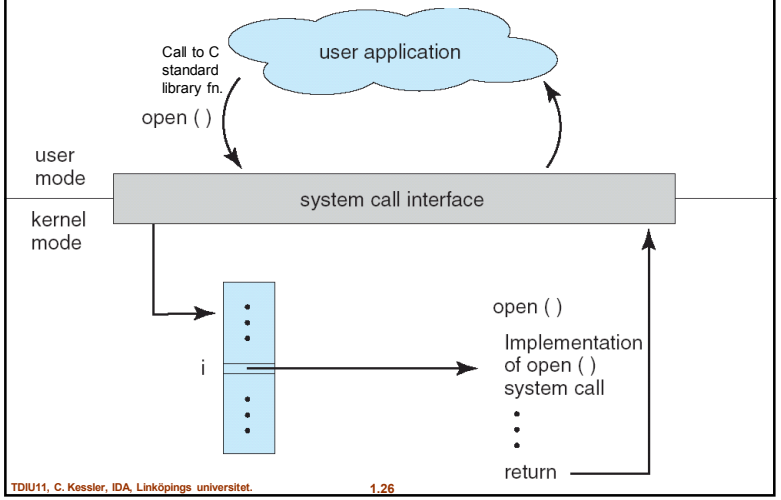
- System call sequence to copy contents of one file to another



Example System Call Sequence

```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```


System Call API – OS Relationship



System Call API Implementation

- **System call implementation** is hardware-specific, e.g. special `trap` instruction with a system call number passed in a register, indexing the interrupt vector (branch table)
- **System call interface** (usually, in C)
  - invokes the intended system call in OS kernel and returns status of the system call and any return values
- Advantage:
  - Caller does not need to know anything about how the system call is implemented
  - Most details of OS interface hidden from programmer by API

Types of System Calls

- **Process control**
  load, execute, end, abort, create, terminate, wait ...
  memory allocation and deallocation
- **File management**
  open, close, create, delete, read, write, get/set attributes...
- **Device management**
  request / release device, read, write, ...
- **Information maintenance**
  get / set time, date, system data, process / file attributes
- **Communications**
  create / delete connection, send, receive, ...

TDIU11
Operating systems

Operating System Operations

Christoph Kessler, IDA,
Linköpings universitet.

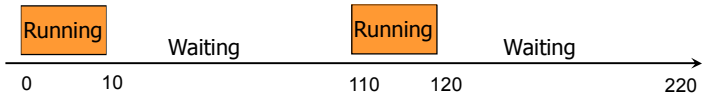
Operating-System Operations

- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap):**
    - Software error (e.g., division by zero)
    - Request for operating system service
- Operating system and users share hardware and software. Make sure that an error in a user program does not cause problems for other programs.
  - Infinite loops, processes modifying each other or the operating system

Operating System Operations

- Dual mode, system calls
- CPU management
  - Uniprogramming, Multiprogramming, Multitasking
  - Process management
- Memory management
- File system and mass storage management
- Protection and security

Uniprogramming

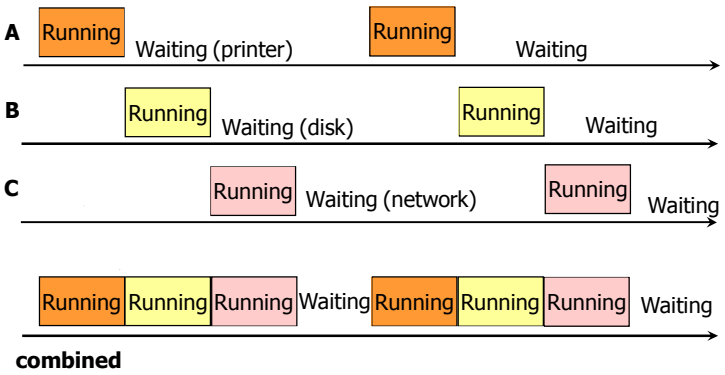


Process execution time:
CPU: 10 + 10 time units
I/O: 100 + 100 time units

I.e., I/O intensive (200/220 = 90.9%), CPU utilization 9.1%

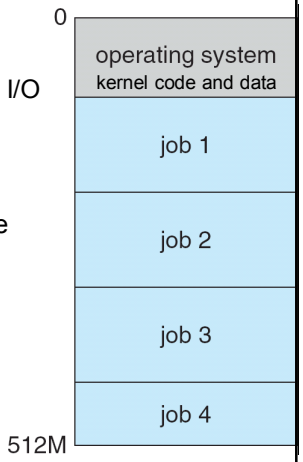
Single user with single program
cannot keep CPU and I/O devices busy at all times.

Multiprogramming with three programs



Multiprogramming

- needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes ***jobs*** (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (e.g., for I/O), OS switches to another job



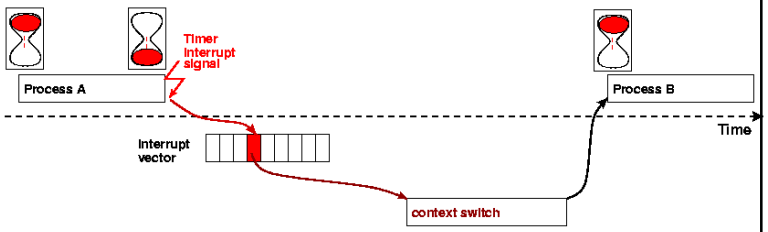
Memory layout for multiprogrammed system

Timesharing (Multitasking)

- Extension of multiprogramming: CPU switches jobs so frequently that users can interact with each job while it is running
  - For **interactive** computer systems, the **response time** should be short (< 1 second)
  - Each user has at least one program executing in memory ⇨ **Processes**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - ⇨ **Virtual memory** allows execution of processes not completely in memory

CPU time sharing using timer interrupt

- **Timer** to prevent infinite loop / process hogging resources
  - Set up to interrupt the computer after specific period
  - System decrements counter at clock ticks
  - When counter = zero, generate an interrupt
  - So, OS regains control and can reschedule or terminate a program that exceeds allotted time



Process Management

- A ***process*** is a program in execution.
  - A unit of work within the system.
  - Program is a *passive entity*, process is an *active entity*.
- Process needs *resources* to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable 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
- Typically, a system has many processes (some user, some system pr.) running ***concurrently*** on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

The operating system is responsible for:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

⇨ Lecture on Processes and Threads

⇨ Lecture on CPU Scheduling

⇨ TDIU16 Lectures on Synchronization

Memory Management

- **Memory**: A large array of words or bytes, each with its own address
  - ***Primary storage*** – directly accessible from CPU
- In order to execute a program, its instructions (or part) must be in memory
- All (or part) of the data that is needed by the program must be in memory.

- **Memory management** determines *what* is in memory *when.*
  - Optimizing CPU utilization and computer response to users

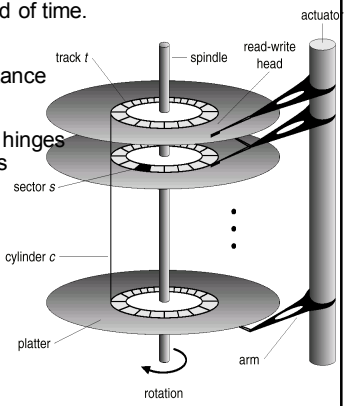
- **OS memory management activities**
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

⇨ Lectures on **Memory Management** and **Virtual Memory**

Mass-Storage Management (1)

- Usually, **disks** are used to store data that do not fit in main memory or data that must be kept for a "long" period of time.
  - ***Secondary storage***
- Proper management is of central importance
- Critical for system performance
  - Often, speed of computer operation hinges on disk subsystem and its algorithms
- **OS activities**:
  - Free-space management
  - Storage allocation
  - Disk scheduling

- Some storage need *not* be fast
  - ***Tertiary storage***
    optical storage, magnetic tape...
  - Still must be managed



Mass-Storage Management (2)

- **OS provides uniform, logical view of information storage**
  - Abstracts from physical to logical storage unit: ***file***
  - Each medium (disk, tape) has different properties:
    access speed, capacity, data transfer rate, sequential/random access
- **OS File-System management**
  - Files usually organized into ***directories***
  - Access control
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files to tertiary storage

⇨ Lecture on **File Systems**

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users,
  to determine who can do what
  - User identities (**user IDs**, security IDs)
  - associated with all files, processes of that user
  - Group identifier (**group ID**)
- **Privilege escalation** allows user to change to effective ID with more rights

⇨ Lectures on **Protection and Security**

System Programs

- provide a convenient environment for program development and execution.
  - File management
  - Status information
  - File modification
  - Programming language support: Compilers, assemblers, debuggers...
  - Program loading and execution
  - Communications: Message passing, e-mail, web browser, ...
- Some of them are simply user interfaces to system calls;
  others are considerably more complex
- Most users' view of the operation system is defined by system programs, not the actual system calls

Summary

- **Operating System** = OS Kernel + System Programs
  - Mediates all accesses to system resources
  - **Interrupt**-driven
    - Error handling
    - Controlled access to system resources, e.g.
      - I/O devices, DMA
      - CPU time sharing
    - …
- **Dual-Mode** (user mode, kernel mode)
  - **System Call** API for portability