Slide 1

Part 1: Introduction

- **What is an Operating System (OS)?**
- Types of Computing Systems
- Computer System Architecture (Review)
- Operating System Services

Slide 2

Is an OS hardware or software ?

What are the goals of an OS?

What is an Operating System?

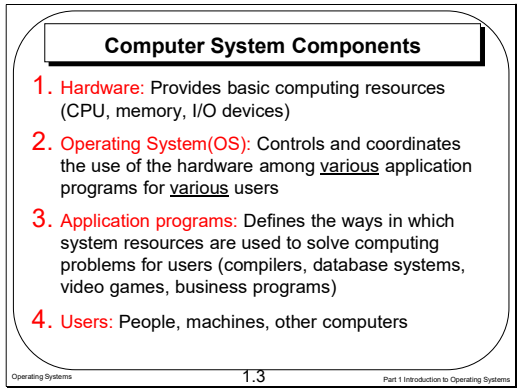
- A **program** that acts as an **intermediary** between a user and the computer hardware → coordinator
- Two major goals: user convenience and efficient hardware utilization
  - Hide hardware complexity → from user program
  - Use hardware in an efficient manner
    - Smart resource (e.g., Central Processing Unit (CPU), I/O devices, memory) allocation
- These two goals can be contradictory
  - Smart resource allocation may require lot of information about user programs → increased complexity

Operating system is a program -> a piece of software. Not hardware. Second, operating system is an intermediate software between user and computer hardware. That means, when users want to run their jobs, their jobs need to go through operating system, and are then executed on the hardware. On the other hand, when computer hardware needs to return the results to the users, the results are first transferred to the operating system, and then back to the users.

The design of operating systems have two goals: user convenience and efficiency.
Convenience: From the user's perspective, the operating system execute user programs and make solving user problems easier, and also makes the computer easier to use.
Efficiency: from the hardware's perspective, the operating system manages the computer hardware in an efficient manner.
These two goals are very important to understand the design decisions in OS. Usually, OS design needs to consider both goals. These two goals sometimes can be in conflict.

Slide 3

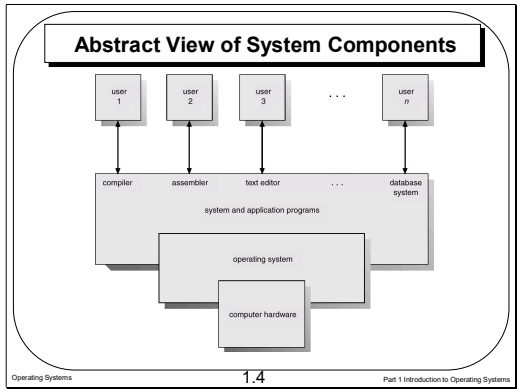
What does the OS do in computer system?



Let's look at the position of an operating system in a computer system. A computer system can be roughly divided into four components: the hardware, the operating system, the application programs and the users.

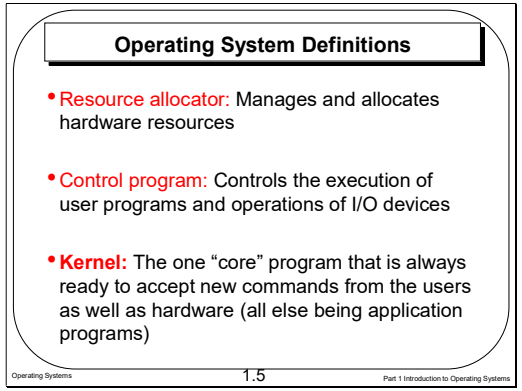
The hardware provides the basic computing resources such as CPU, memory, and I/O devices. Computers can have quite different hardware, as we will see in the computing history.
The application programs define the ways how the system resources are used to solve the problems. Example application programs include word processors, spreadsheets, video games, etc.
The users can be people, or machines, or other computers in a networked system.
Among hardware, application programs, and users, there is a gap: how to control the resource usage among different application programs for different users. This is the role of the operating system. The operating system controls and coordinates the use of the hardware among the various application programs for different users.

Slide 4



Slide 5

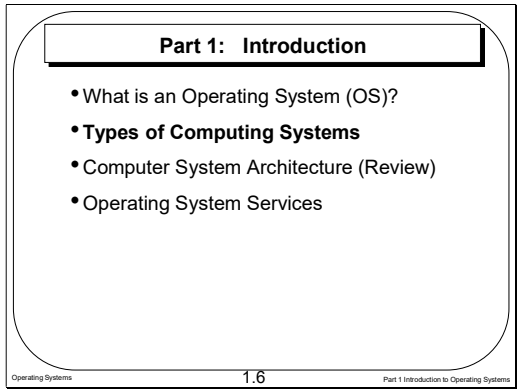
What are the 3 definitions of an OS? Explain.



Based on the abstract view, we can see what an operating system does.

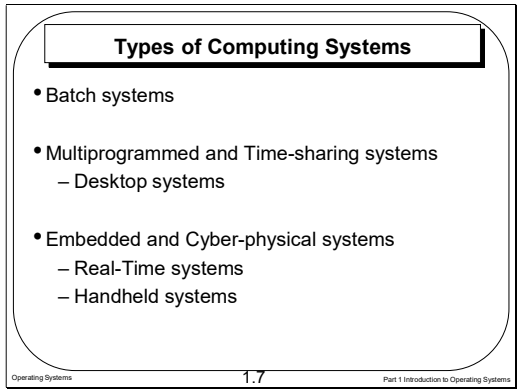
The operating system is a resource allocator. It allocates resources to different application programs.
The operating system is a control program. It controls the execution of user programs and operations on devices.
We also call operating system Kernel. It is the one "core" program that is always ready to accept new requests from users as well as the hardware.

Slide 6



Slide 7

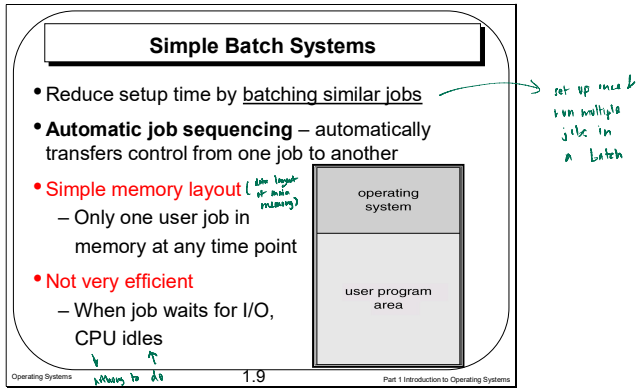
Time-sharing system
- time slice your CPU resources
→ can set priority, etc.
→ structure will switch between programs periodically so that they can share the hardware



Now, we give some examples on different types of operating systems. We will introduce them one by one.

Slide 8

Why does the operator often batch jobs with similar needs?
What is automatic job sequencing?
What is the disadvantage of a simple batch system?



In the first generation of computing systems, each job has a setup cost, for example, loading the data into the main memory and figure out the memory layout. To reduce the setup cost, the operator often batch the jobs with similar needs. We set up once and runs multiple jobs in a batch.

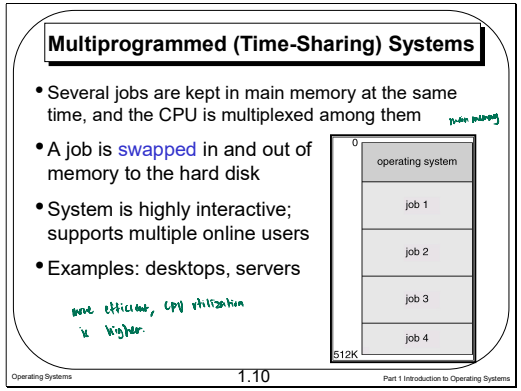
The batch system needs to figure out which job to run in the batch. It has a functionality called automatic job sequencing, order the jobs from one job to another automatically.

Finally, the batch system initially has very simple memory layout (or the data layout in the main memory). Only allow one user program in the main memory. In this figure, the memory is divided into two parts: operating system and the one for user program, only one user program.

What is the problem for simple batch system? A job usually has I/O and CPU computation. Imagine the job is waiting for I/O. The major problem of running a job at one time is efficiency. When a job is waiting for I/O, the CPU has nothing to do, and become idle.

Slide 9

Why are multiprogrammed systems more advanced than simple batch systems?



The CPU is multiplexed among the jobs. When a job is waiting for I/O, we will give the CPU to other jobs. For example, we can allow job 1 to run some time and then switch to job 2. Job 2 takes the CPU for some time. later Job 3 and so on.

The memory layout is different from the simple batch system. As you can see, the program data of multiple jobs are stored in the main memory at the same time, instead of a single job in simple batch system.

One question: why are multiprogrammed systems more advanced than simple batch systems? Efficiency. In multiprogrammed systems, the CPU utilization is higher.

Time sharing systems are a logical extension from multiprogrammed systems. In a time sharing system, the CPU executes multiple jobs by switching from one to another. The switching is so frequent that the users can interact with the system while the other jobs are running.

There are two unique features that are beyond multiprogrammed systems of earlier generations:

In order to support a larger number of jobs, the time sharing systems can swap the job in and out of memory to the disk. This is called the functionality of virtual memory. We will discuss the concept of virtual memory in details in the future lectures.

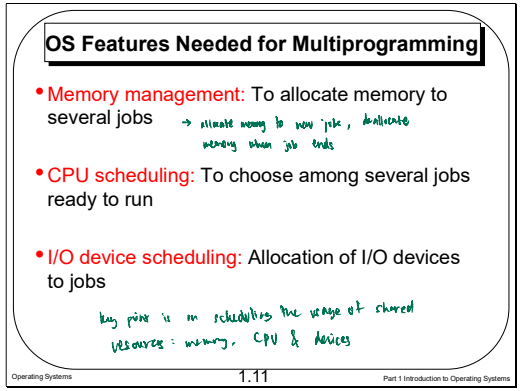
Another feature that are stronger than the batch system is user interaction. The time sharing system allows users to input the command, and get the command running in an online manner. The system needs to be interactive, since the user submits the command and waits for the results.

**Time sharing is a kind of multi-programmed systems. There are many forms of multi-programmed systems. Nowadays, almost every OS that you see is a multiprogramming system.

There is no such a strict definition for time sharing systems. However, in my course, we consider time sharing systems are a natural extension to initial multiprogramming systems, with focus on virtual memory support and user experience.

Slide 10

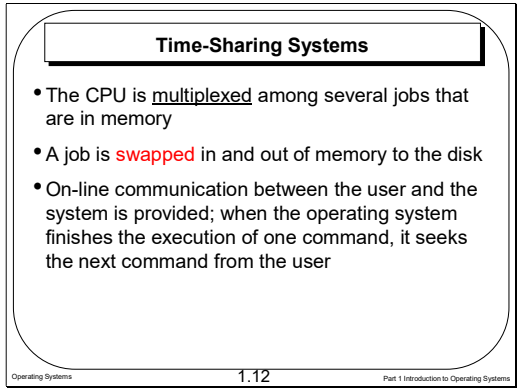
In order to support multiprogramming, what features should the operating system provide?



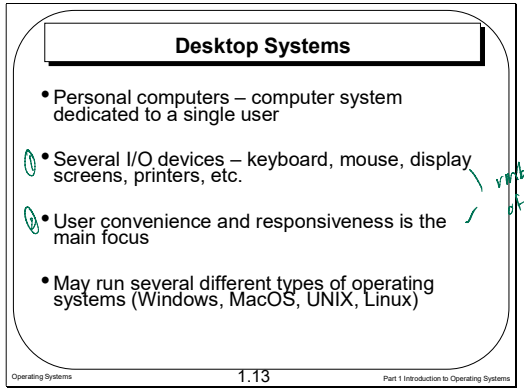
In order to support multiprogramming, what features should the operating system provide? The key point is on scheduling the usage of shared resources: memory, CPU and devices.

First, memory management. The operating system should have the capability of allocating memory to multiple jobs. We need to allocate the memory for new jobs, and deallocate the memory when a job ends.
The second feature is to decide which job to run. There are many criteria for scheduling. We discuss the topic of CPU scheduling in a later lecture.
Third, the operating system should know how to allocate the devices to jobs, since the devices are now shared by multiple jobs.

Slide 11

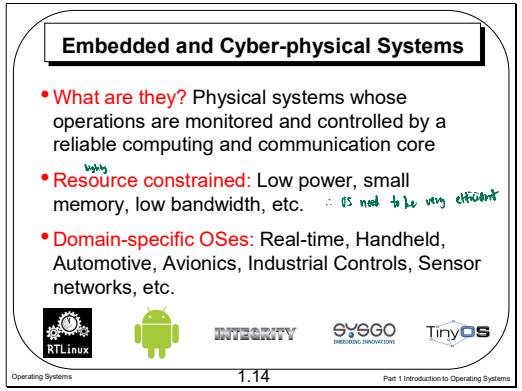


Slide 12



The batch and time sharing systems are already far away from us. We are already familiar with Windows, MacOS, Unix, Linux etc. Every day, we are using them. I want to highlight two points here:
First, each PC has many I/O devices: keyboards, mouse, small printer etc. this is different from the traditional/old systems. What is the impact? Hardware driver can be a huge part in the code base.
Second, user convenience and responsiveness is the top goal. Remember that we have two goals for operating system: efficiency and user convenience. Here, desktop systems favor user convenience, for example, desktop systems can have very fancy GUI. Traditional/old systems typically focused on efficiency.

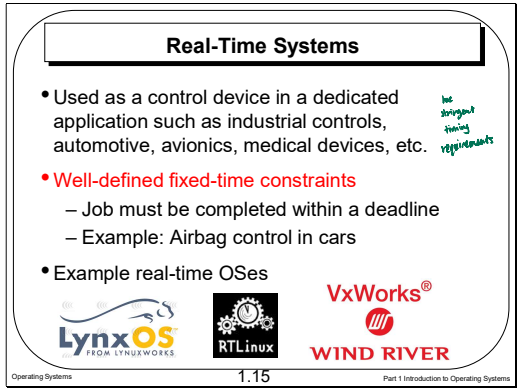
Slide 13



Cyber-physical systems are typically embedded in another physical system that they monitor and control. Examples include automotive, avionics, medical devices, industrial controls, Internet-of-Things (IoT), smart home devices, future mobility devices, etc.

These systems are highly resource constrained, and hence the operating system is required to be very efficient. In some cases, user responsiveness is also important, for example in the case of handheld devices.

Slide 14

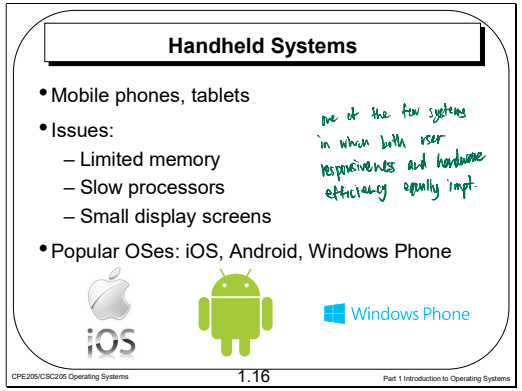


Real-time systems have well defined fixed-time constraints on the job execution. Operations must be finished within the defined time constraints, otherwise the system will fail. Widely used as a control device in special applications. For example, some scientific experiments, automotive, avionics, industrial control and display systems.

Real-time systems are generally a subset of embedded and cyber-physical systems; those embedded systems that have stringent timing requirements.

Some popular Oses are given here. you may not be familiar with those names. The e-learning lecture (week 7) is about real-time OS. That can help you to know more about those operating systems.

Slide 15

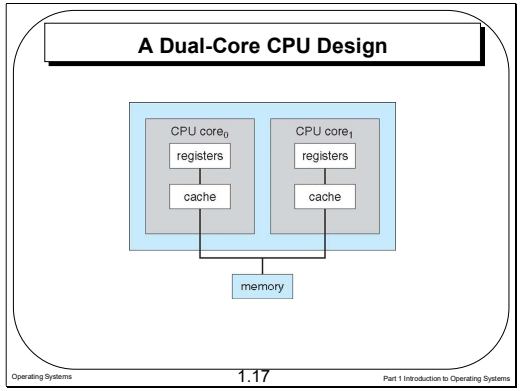


They are widely used in mobile phones and pads. They can be regarded as a subset of embedded and cyber-physical systems. The popular handheld operating systems include iOS and Android. Compared with the desktop operating systems, the handheld devices have some special issues:

-Memory: Unlike the PC with several GB of main memory, the handheld devices have limited memory.
-The second issue is that the processor speed is slow.
-Finally, the display screens are small. The system needs to do some clipping in order to fit the content to the display.

One of the few systems in which both user responsiveness and hardware efficiency are both equally important.

Slide 16

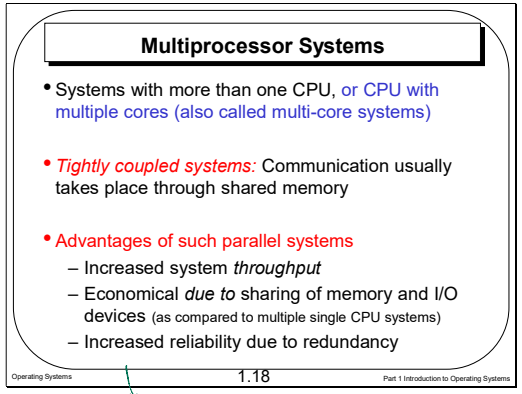


Operating system development can be also driven by computer hardware. Recently, we have witnessed that the CPUs can have multiple cores. For example, in this figure, we have a CPU with two cores. The two cores share the main memory. Each core has its local registers and caches.

As the CPU has multiple cores, how the operating system is adapted to the multi-core design?

Slide 17

multi-programming: system can run multiple programs
multi-processing: system can run on multiple CPUs or CPU with multiple cores.



system have reliable. failure of one processor will not halt the entire system.

Multiprocessor systems are targeted towards hardware with more than one CPU, or CPU with multiple cores.

Multiprocessor systems are also known as tightly coupled systems: communication is performed through shared memory. In comparison, loosely coupled systems are OSes running on distributed computers. You will learn them in other courses at higher level.

Multi-programming: the system can run multiple programs.
Multi-processing: the system can run on multiple CPUs or CPU with multiple cores.

There are multiple advantages of using such parallel systems:
-First, of course, it can increase the system throughput.
-The second one is economy. Memory and I/O components are shared.
-Finally, the system is more reliable. Failure of one processor will not halt the entire system.

Slide 18

Part 1: Introduction

- What is an Operating System (OS)?
- Types of Computing Systems
- **Computer System Architecture (Review)**
- Operating System Services

Slide 19

Computer System Architecture

- Computer System Operation
- Storage Hierarchy
- Hardware Protection

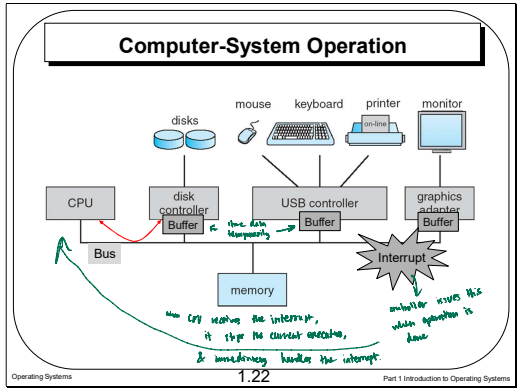
Slide 20

Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Device controller moves data between local buffer and memory
- Device controller informs CPU that it has finished its operation by causing an *interrupt*

The following concepts are important in computer system operation.

Slide 21



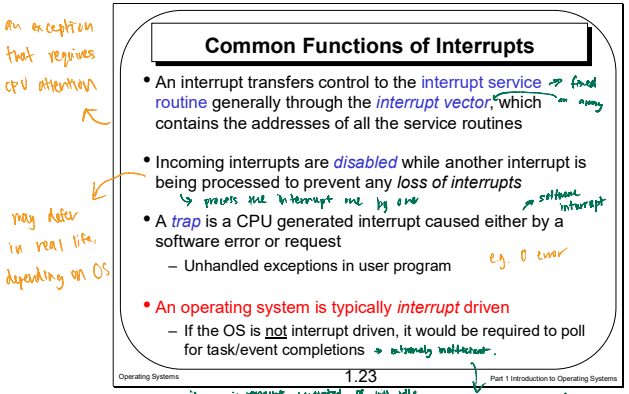
A modern computer system consists of a CPU and a number of devices , for example, disks, printer etc. these devices are usually connected with a common system bus.

A device controller is in charge of the operations of a specific type of device. The CPU and the device controllers can execute concurrently, and compete for the bus cycles. The device controller usually has a local buffer. The buffer is used to store the data temporarily.

After the CPU issues an I/O instruction, how does the CPU know the operation is done or not? in operating system, this event is called interrupt. The device controller issues an interrupt to the CPU when the operation is done. In hardware, it is an electronic signal via the CPU interrupt pin. When the CPU receives the interrupt, it stops the current execution, and immediately handles the interrupt.

You can see there are other variations in the computer system architecture, for example, multiple CPUs and dedicated bus for device. But the basic concepts of this slide still apply.

Slide 22



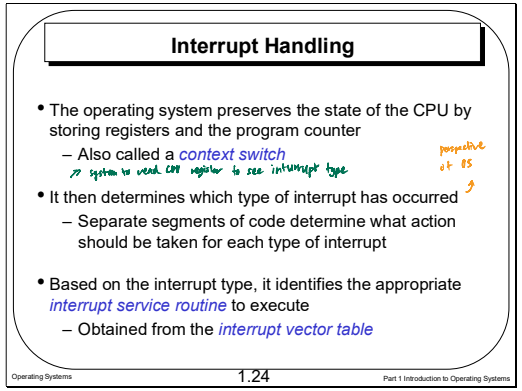
Interrupt is a very important concept in OS. Different operating systems have different interrupt handling mechanisms. But there are multiple common functions for interrupt handling. First, all the interrupt handling routines are fixed, and their addresses are stored in an array called interrupt vector.

Second, to prevent a lost interrupt, the incoming interrupts are disabled while another interrupt is being handled. That means, we process the interrupt one by one.

Third, the interrupt can be generated from hardware or software. A software generated interrupt is called a trap. Usually, the software has an error (or exception) or issue a request to the operating system service (via system call). For example, an exception in a java program, or an unhandled exception in a user program.

Finally, all operating systems are interrupt driven. That means, if there is no interrupt generated, OS will idle. One more issue to consider: if the OS is not interrupt driven, it would be required to poll for the task/event completion from time to time. That is extremely inefficient. Interrupt is more efficient than polling.

Slide 23



There are multiple steps for handling an interrupt.

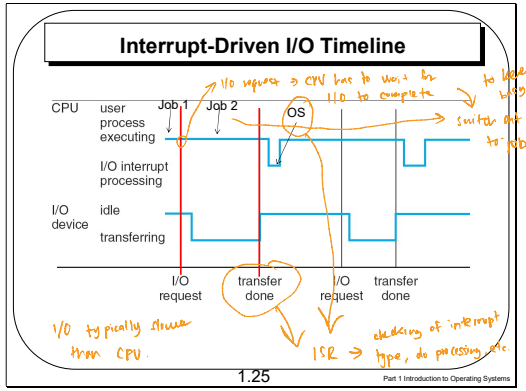
First, the operating system must save the state of the current execution: the address of the next instruction (the program counter), the state of the CPU including the registers. You will know more details when we talk about the concept of processes and context switches.

Second, we determine which type of interrupt has occurred. The system needs to read the CPU register to see the interrupt type.

Third, the interrupt service routine is executed according to the type of interrupt. How can we find the target interrupt service routine? Interrupt vector.

One typical usage of interrupt is interrupt I/O, usually for low-speed I/O devices.

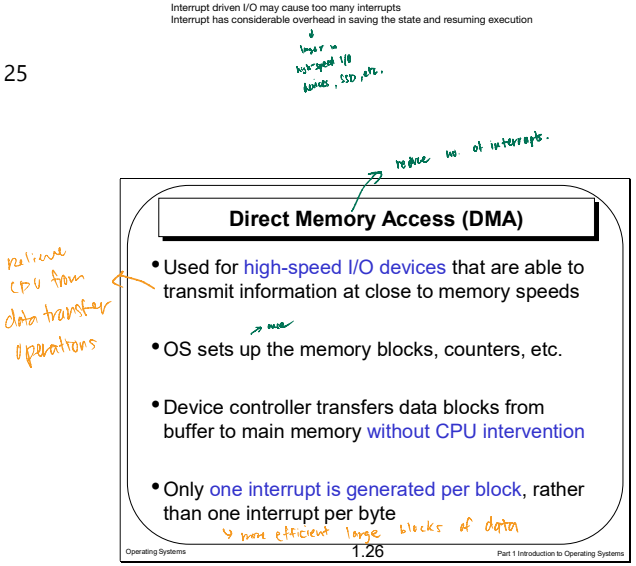
Slide 24



Let's see one example for interrupt handling. In this figure, we have the timeline of interrupt handling.
We consider two states for CPU: executing the user process or handling I/O interrupt
Two states for I/O device: idle (nothing to do) or transferring data.

Initially, the I/O device is idle and the CPU is executing the user process, say Job 1. Now, an I/O request is issued, and the CPU continues the execution for another job, say Job 2 (recall multiprogrammed system). After some time, the I/O completes, and an interrupt is issued to the CPU. The disk becomes idle. Then, CPU starts the I/O interrupt handling, that is, to execute the interrupt service routine in OS. After that, the CPU resumes the execution of user process. And so on and so on. You can see that I/O and CPU execution happen at the same time.

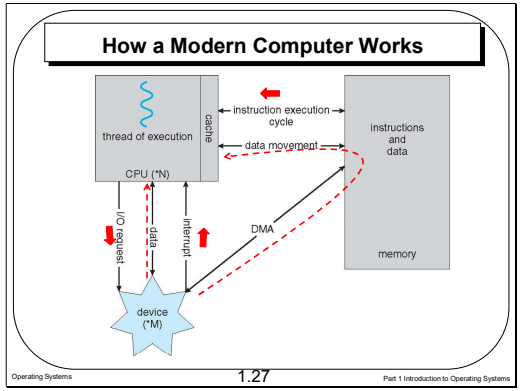
Slide 25



Interrupt driven I/O may cause too many interrupts. Consider that you want to copy 1 TB of data, and each byte causes an interrupt in the worst case. On the other hand, interrupt has considerable overhead in saving the state and resuming execution. The overhead can be larger for high-speed I/O devices such as solid state drives. There must be some mechanism to reduce the number of interrupts. Direct memory access, DMA, is a common technique.

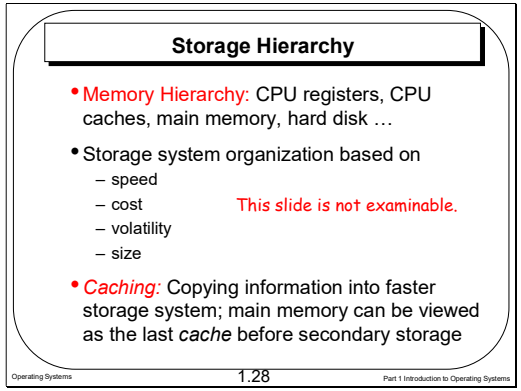
In DMA, the operating system sets up the buffer, pointers, and counters once. And then, the device controller transfers blocks of data from the buffer directly to the memory without CPU intervention. Therefore, only one interrupt is generated per block, and the CPU is free to perform other tasks.

Slide 26



With the concept of interrupt, let's see how a modern computer works. We have three parties: CPU, memory, and device. The memory stores the instructions and data. A job execution fetches instructions from the memory, and read/write data from/to the memory. After the CPU issues an I/O request to the device, the device will notify the CPU when the data is ready. Depending on whether interrupt driven I/O or DMA is used, the data buffer location is different. If DMA is used, the data is stored in the memory. Otherwise, the CPU can get the data directly from the device buffer.

Slide 27



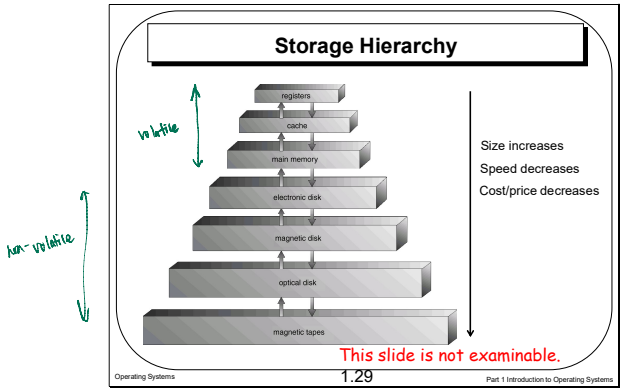
In our last lecture, we stopped here. First of all, let's review our previous lecture. We started with multiprogramming systems. Multiprogramming means that the system can run multiple jobs at the same time. We then talk about multi-processing systems. Multiprocessing means the system can run on multiple CPUs or CPU with multiple cores. Then, we learnt one important concept: interrupt. About the interrupt, you should know about the steps on how OS handles an interrupt. Also, the difference between interrupt driven I/O and DMA.

We now have a quick introduction on memory hierarchy. The modern computer system has a storage hierarchy: CPU registers, CPU caches, main memory, hard disk.

Volatile memory means, once the power is turned off the data is lost. Non-volatile memory means, the data is there even after the power is turned off.

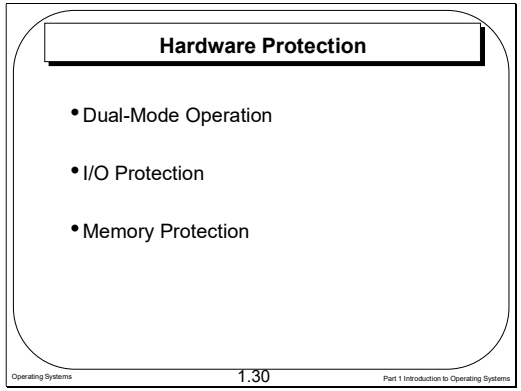
One key concept in the storage hierarchy is caching. We copy the frequently accessed data to faster storage and hope that the CPU accesses that data again in the near future. Memory can be considered as the last cache before secondary storage. Consider how CPU accesses the data. First on the register. If there, then return. Otherwise, look for it in the cache. If the data is found in the cache, then return. Otherwise, look in main memory. There is a rule in computer systems: locality, also called 80/20 rule, which states that 80% of memory accesses go to 20% of data items. That's why caching is effective for computing systems.

Slide 28



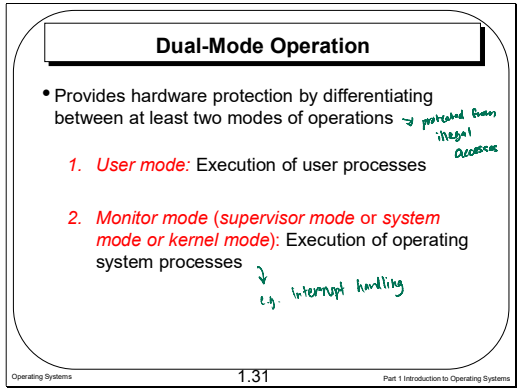
This figure shows the storage device hierarchy in modern computers. From top down, we have registers, cache, memory, disk, tapes etc. From top down, the cost decreases, the speed also decreases, the size increases. The levels above main memory (including main memory itself) are volatile; the layer below is non-volatile.

Slide 29



Hardware resources need to be protected. Recall that OS must control and coordinate the access to hardware. However, in practice, a malfunctioning program may issue illegal I/O instructions, accessing memory locations of the operating system. The operating system should provide protections in hardware to avoid such scenarios. So, we will first introduce the concept of dual-mode operation, and then, see how the concept of dual-mode operation can be used to implement I/O and memory protection.

Slide 30

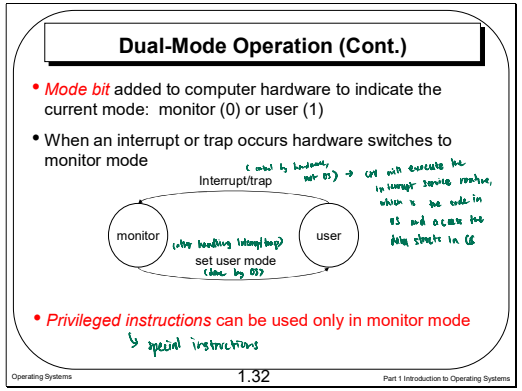


Because there can be illegal accesses, for example, virus from applications, we should distinguish the execution whether it is simple normal execution from user process, or whether the execution will access the hardware or other important data structures in OS. In the latter case, we must make sure the execution does not issue illegal operations to hardware and data structures of OS.

In the user mode, we execute user processes; in the monitor mode, we execute the operating system processes (or kernel). Interrupt handling is an example of operating system process.

Slide 31

What are privileged instructions? Can they be used in user mode?

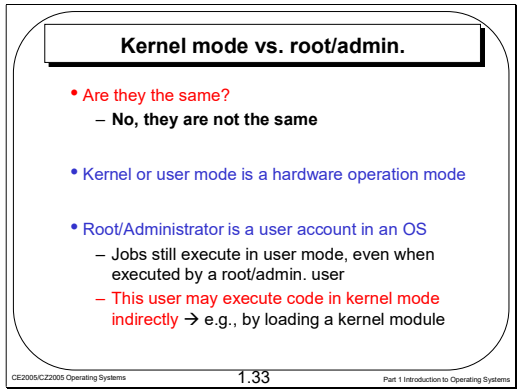


Modern hardware supports a bit named mode bit to indicate the two modes of operations. A bit value of zero means monitor mode; a bit value of one means user mode.

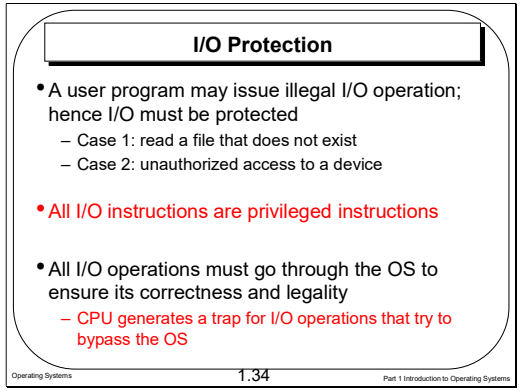
Let's see how the system transitions between these two modes. Suppose we are in the user mode, executing the user process. When a trap or interrupt occurs, the system will switch to the monitor mode automatically (controlled by hardware and not OS). This is because the CPU will execute the interrupt service routine, which is the code in operating system and access the data structures in OS. After handling the trap or the interrupt, the system switches to the user mode. This switch back to the user mode is controlled by the OS.

There are some special instructions that are only allowed to be executed in the monitor mode. These instructions are called privileged instructions. In the following, we will see how to use the concept of privileged instructions to support hardware protections for I/O and memory.

Slide 32



Slide 33



Suppose we have a user program with computation and I/O. The normal computations are in the user mode. After some time, the process issues an I/O request and the process waits for I/O to complete. After the I/O is done, the interrupt service routine is executed. That is the OS code, and thus, it is in monitor mode. After the interrupt service routine is done, we resume the computation in the user process.

Slide 34

the functionality of memory protection is to protect the memory space of a process from being modified by other processes

important for multiprogram systems

Memory Protection

- Must provide memory protection, at least for the interrupt vector and the interrupt service routines
  - using the memory area for each program
- Two CPU registers determine the range of legal addresses a program may access:
  - **Base register**: Holds the first legal physical memory address → starting address
  - **Limit register**: Contains the size of legal range
- Memory outside the defined range is protected and cannot be accessed
  - illegal

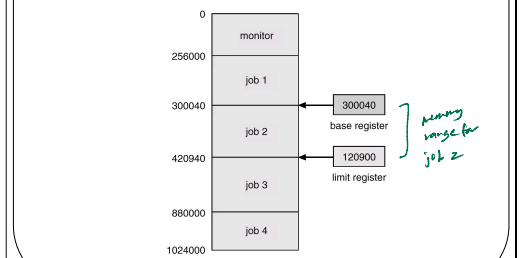
We need to protect the memory in many ways. For example, we must protect the interrupt vector and the interrupt service routines in OS. If those data and instructions are changed, the operating system will go wrong. For example, a virus can intentionally change those data structures to do the wrong things.

The functionality of memory protection is to protect the memory space of a process from being modified by other processes.

The basic idea is to define the memory area for each program. In particular, add two special registers that determine the range of the memory addesses that a program can access: the base and the limit registers. The base register holds the starting address, the limit holds the range. That means, a job can only access the memory region defined in base and limit. Accesses to the memory outside this range is considered to be illegal.

Slide 35

Memory Protection (Cont.)

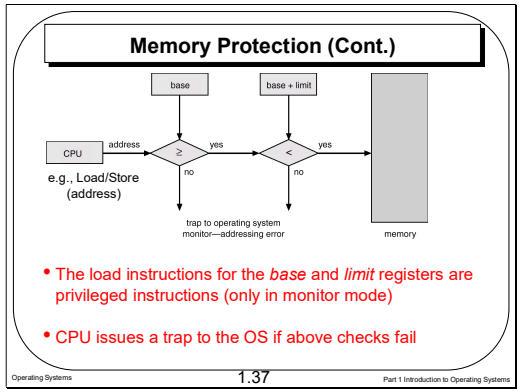


Let's look at one example of memory protection. We consider this memory layout. Btw, is it a simple batch system? No. it is multiprogrammed system.
Here, the base is 300040, the limit is 120900. therefore, the memory range for job 2 is from 300040 to 420940.

Is the above diagram a simple batch system? If not, what is it?

Why the loading instructions for base & limit are privileged instructions?

Slide 36

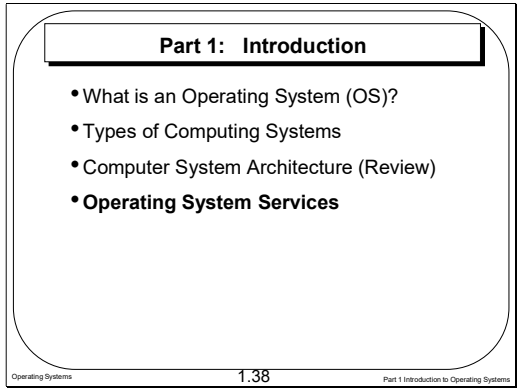


Ans to qns: loading means loading the values from main memory to CPU registers. OS controls the value of base and base + limit, not the application itself.

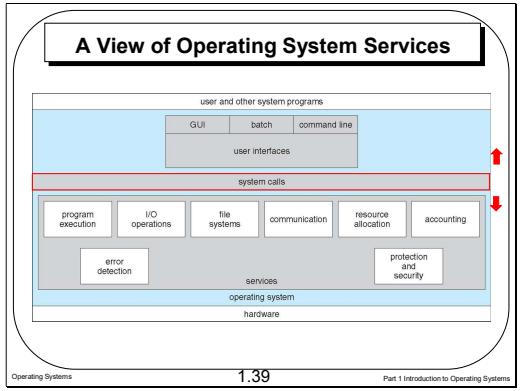
let's see how the memory protection is performed. Given a memory address to access, we first check whether it is larger than the base register. If failed, a trap is issued to the operating system. This addressing error is a segmentation fault in Linux. Then check whether the address is smaller than the sum of (base and limit) registers. Again, if failed, a trap is generated. If both tests are passed, the memory access is issued to the main memory.

Why the loading instructions for base and limit are privileged instructions? loading means loading the values from main memory to CPU registers. OS controls the value of base and base+limit, not the application itself.

Slide 37



Slide 38



We will have a quick introduction on operating system services. In this figure, we look into the internals of an operating system. Operating system has multiple layers: user interfaces, system calls, and services. To allow users to access the operating system services, a set of API is offered. Those APIs are called system calls. The services include program execution, file system etc. Again, if you are puzzled about some terms, don't worry, we will introduce them in later lectures.

One set of operating-system services provides functions that are helpful to the user:

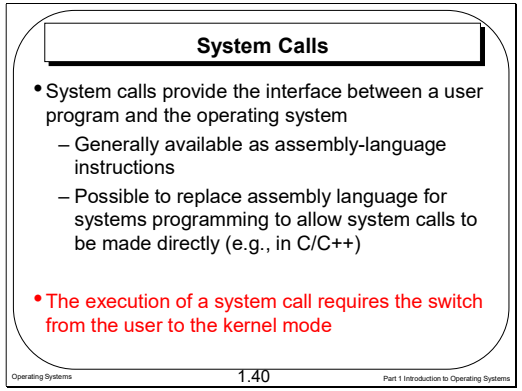
User interface - Almost all operating systems have a user interface (UI). Varies between Command-Line (CLI), Graphical User Interface (GUI), Batch, etc.

Program execution - The system must be able to load a program into memory and run that program, and also end execution either normally or abnormally (indicating error).

I/O operations - A running program may require I/O, which may involve a file or an I/O device.

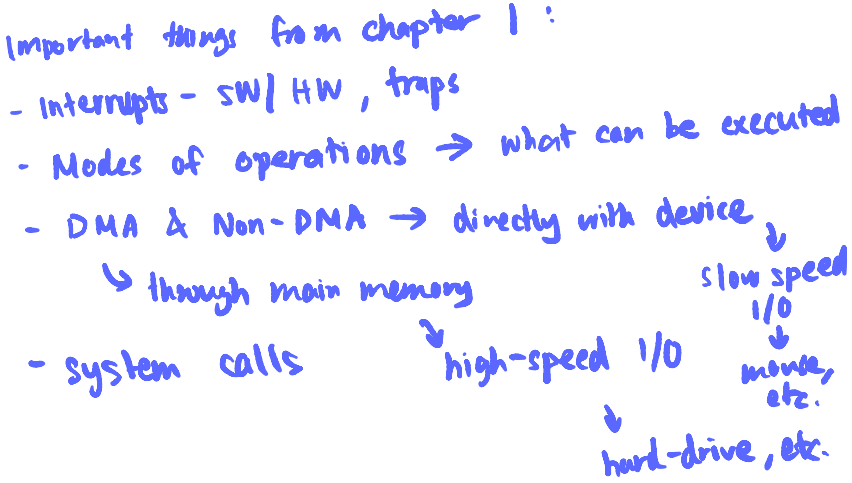
File-system manipulation - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management, etc.

Slide 39



System calls are generally available as assembly language instructions, and now system calls in high level languages such as C is also available. For example, those system calls in UNIX and Windows can be directly invoked in C and C++.

The execution of a system call requires the switch from the user to the kernel mode.



Slide 40

Why must the mode switch from user to kernel before system call is executed?

How many mode switches are done in the diagram?

Describe what the system call interface do.

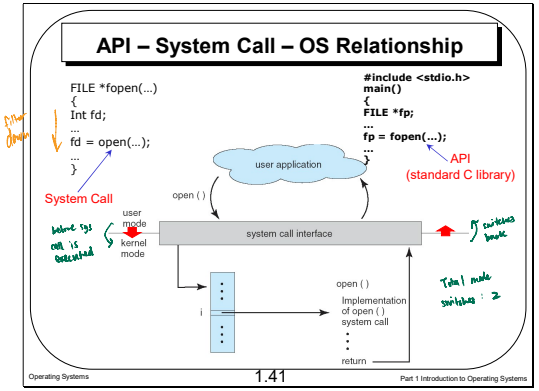
Caller need not know anything about how system call is implemented

just needs to obey API & understand what OS will do as a result of the call

System call interface → maintains a table indexed according to these numbers.

→ invokes intended system call in OS kernel & returns status of system call and any return values.

All I/O instructions are privileged instructions → must be executed in kernel mode



Let's see one example on the details of executing a system call. This is a C program, which opens a file. It uses fopen(). It is similar to opening a file in java. The implementation actually consists of a system call, open. Now, before the system call is executed, we change the mode from user to kernel. Why? All the I/O instructions are privileged instructions, and they must be executed in the kernel mode. Then, we allocate the instructions of open() and execute the open() system call. When it is done, we need to get back to user mode and continue the execution of user process. In summary, how many mode switches do we have? This is an exam question a few years ago. Two.

Typically, a number is associated with each system call. System-call interface maintains a table indexed according to these numbers. The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values.
The caller need not know anything about how the system call is implemented. He just needs to obey API and understand what OS will do as a result of the call.

Most details of the OS interface are hidden from programmer by API. They are managed by run-time support libraries (set of functions built into libraries included with compiler).