University of Nottingham
UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 1: Introduction

University of Nottingham, Ningbo China
2024

Module Convenors

- Week 2 – Week 9
  - Name: Dr. Fiseha Berhanu Tesema
  - Office: PMB-442 (by appointment)
  - Office Hour: Moodle Page
  - E-mail: Fiseha-Berhanu.Tesema@nottingham.edu.cn

- Week 10 – Week 14
  - Name: Dr Qian Zhang
  - Office: SEB-438, Office hour (physical)
  - Office Hour: Moodle Page
  - E-mail: qian.zhang@nottingham.edu.cn

Goals
What and How?

- Goals:
  - Introduction to the **fundamental concepts**, **key principles** and **internals** of operating systems and concurrency.
  - Better understand how **application programs interact/rely with the operating system**
  - Basic understanding of writing **concurrent/parallel code.**
- How:
  - Through **lectures** (2hrs/w for 11 weeks, 3hrs/w for one week(Week 14th))
    - Monday, DB-C05, 15:00 – 17:00 (2 hour session)
    - Tuesday, DB-C05, 13:00-14:00 (1 hour session)

  - Through **labs/coursework** (lab starts from 25th September)
    - Wednesday, IAMET-406, 11:00-13:00 (2 hour session)
    - Students from both sessions arrive at 11:00, take attendance according to your timetable
    - There will be in-lab quizzes during the 1st, 6th, and 12th week. These quizzes are worth 0%, 2%, and 3% respectively, contributing to your total mark.

Goals

What and How?

- **Lectures** will **introduce** the concepts
- The **labs** will teach you:
  - Practical experience/insights in fundamental **OS concepts**
  - The use of operating system APIs and implementation of OS **schedulers**
  - The basics of **concurrency**
  - Help you with practical **implementations/coding** (e.g., concurrent programs)

Assessment

What Should You Expect

- A **120 minute exam** that focusses on:
  - **Knowledge**
  - **Comprehension**
  - **Application**
- The exam will be 3 out of 4 questions, with 75% of the assessment on the exam.
- **Sample questions** from previous years are available on Moodle and will be included in the lectures
- The **coursework** is worth 20%
  - Coursework uses the **concepts introduced in the labs**
- **In-Lab Quiz** is worth 5%

Content

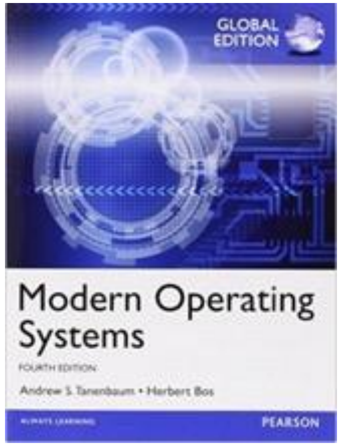
Subjects We Will Discuss

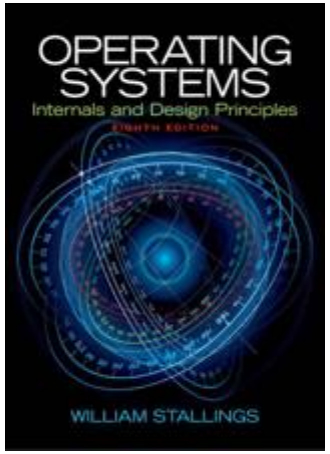
Subject	# Lectures
Introduction to operating systems/computer design	**2**
Processes, process scheduling, threading, ...	**4-5**
Concurrency, deadlock	4-6,2
Memory management, swapping, virtual memory,…	**6-7**
File System, file structure, management,…	4-5
Revision	1

Table: Preliminary course structure

Reading Material

- Books:
  - Tanenbaum, Andrew S. 2014 **Modern Operating Systems.** 4th ed. Prentice Hall Press, Upper Saddle River, NJ, USA.
  - Silberschatz, Abraham, Peter Baer Galvin, Greg Gagne. 2008. Operating System Concepts. 8th ed. Wiley Publishing.
  - Stallings, William. 2008. **Operating Systems: Internals and Design Principles**. 8th ed. Prentice Hall Press, Upper Saddle River, NJ, USA.
  - Thomas Anderson and Michael Dahlin. 2014 Operating Systems, Principles & Practice. 2nd Ed. Recursive Books, Ltd.





- Other sources:
  - Daniel P. Bovet, Marco Cesati Understanding the Linux Kernel. 3rd ed. O’Reilly Media, November 2005
  - **Course slides** will be available on Moodle

University of Nottingham
UK | CHINA | MALAYSIA

Goals for Today

Overview

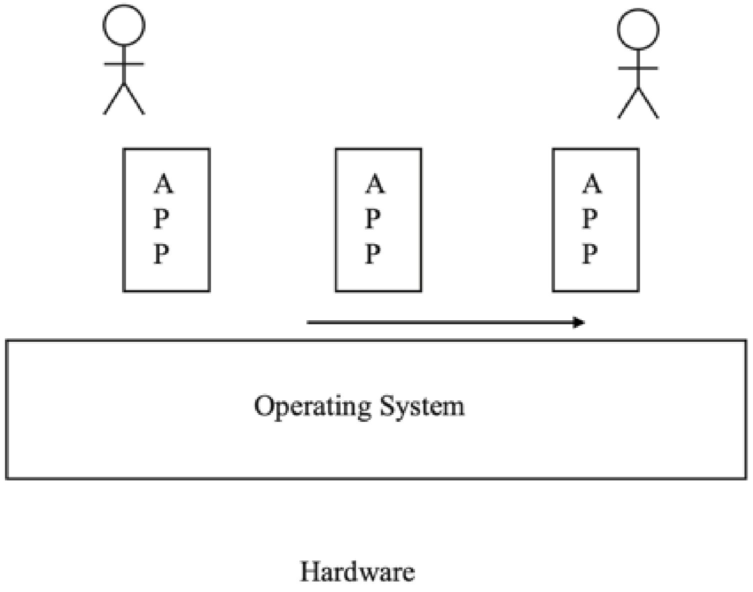
- **Defining** operating system
- What is **multi-programming**
- **Kernel-user mode** and **system calls**

What is Operating System

- Modern computer consists of processor, main memory, disks, printers, keyboard, mouse, display and various other input/output (I/O) devices.

- Operating system is a layer of software to provide user programs with a **better**, **simpler**, **cleaner** model of the computer and to handle **managing all the resources**.

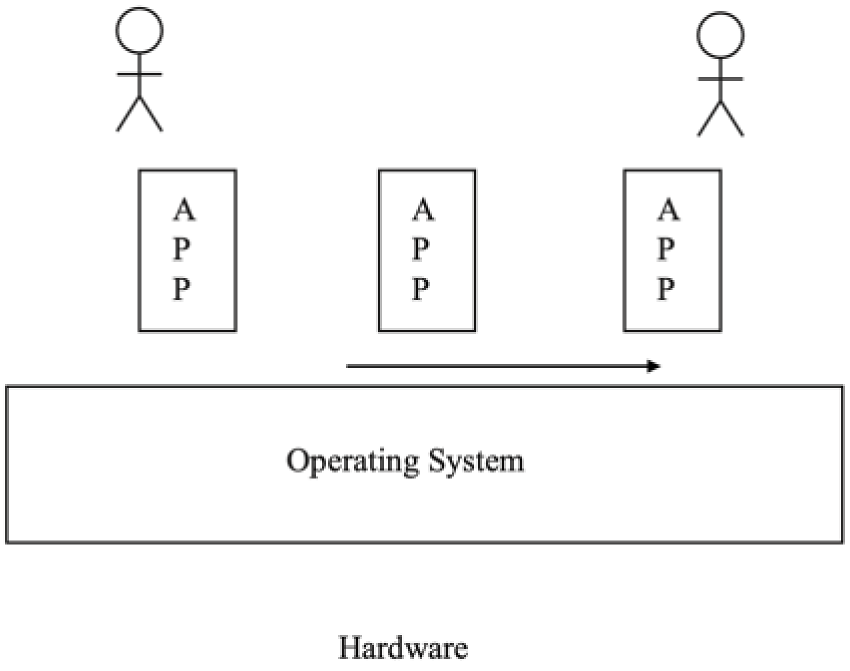
- User normally interact with it using user interface programs like Shell and GUI (Graphical User Interface)



What is Operating System

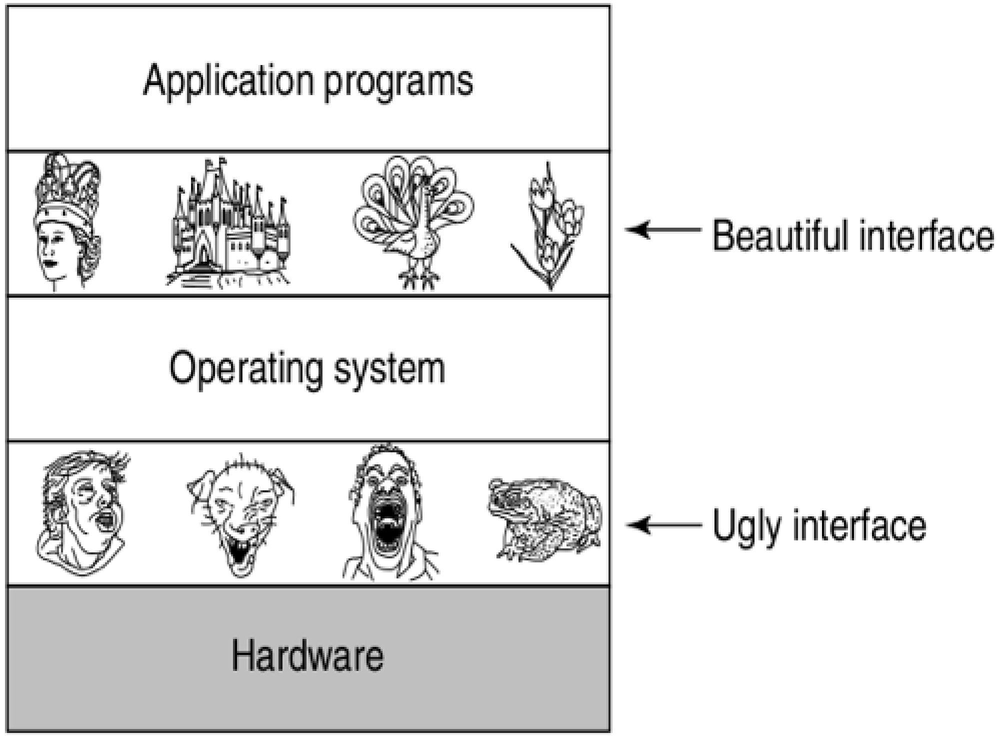
- For Application Programmers
  - It is an **extended machine**
    - Hides the messy details which must be performed
    - Presents user with a **virtual machine**, easier to use

- For OS Kernel Developers
  - It is a **resource manager**
    - Each program gets time with the resource
    - Each program gets space of the resource



Defining Operating Systems

- An operating system is a layer of indirection on top of the hardware:
  - It provide **abstractions** for application programs (e.g., **file systems, process, address space**)
  - It provides a **cleaner** and **easier interface** to the hardware and hides the complexity of ‘bare metal’
  - It allows the programmer to be lazy by using common routines



What is Operating System

What can an OS do for me?

- Where is the file physically written on the disk and how is it retrieved (file systems)

- Why do instructions look the same regardless of the device? (abstraction)

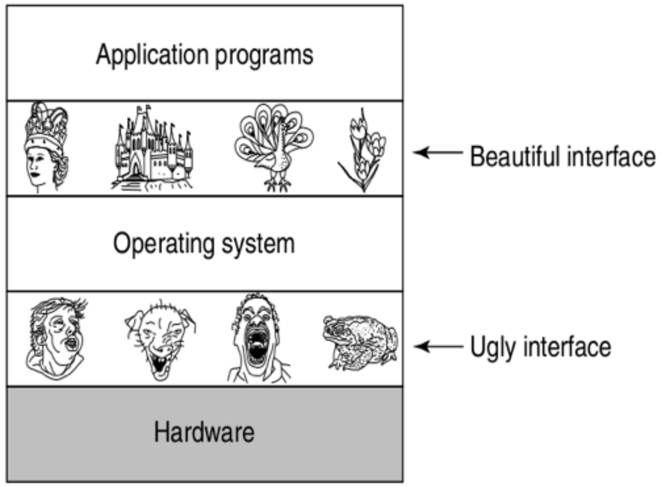
- What if multiple programs access the same file simultaneously?(processes, concurrency,…)

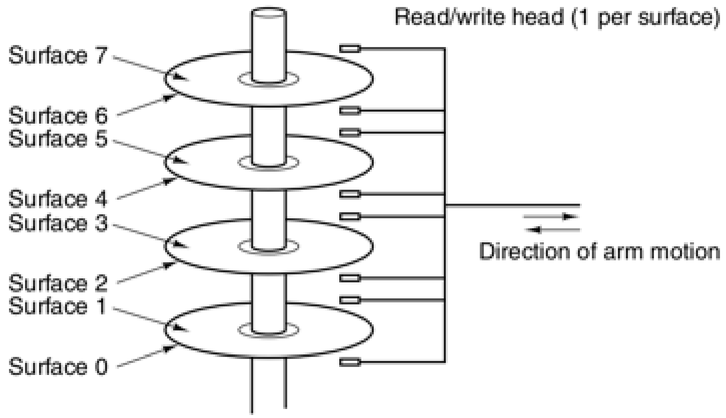
- Why is the access denied? (protection)

Defining Operating Systems

A virtual Machine Providing Abstractions

- **Challenges of Early Programming:**
  - Programmers had to manually control each piece of hardware, such as memory, input/output devices, and the CPU, often needing to know the specific characteristics of the hardware they were working with.
  - **(Ugly Hardware)** Real computer hardware is intricate, inconsistent, and difficult to control. It requires precise timing, memory management, and handling of peripheral devices like keyboards, printers, and displays.





Defining Operating Systems

A virtual Machine Providing Abstractions

- **Challenges of Early Programming:**
  - Adding Two Numbers in Assembly (x86 architecture)

```
MOV AX, [1000h]  ; Load the value from memory address 1000h into register AX
MOV BX, [1001h]  ; Load the value from memory address 1001h into register BX
ADD AX, BX       ; Add the values in AX and BX, store the result in AX
MOV [1002h], AX  ; Store the result from AX into memory address 1002h
```

How This Relates to Early Programming:

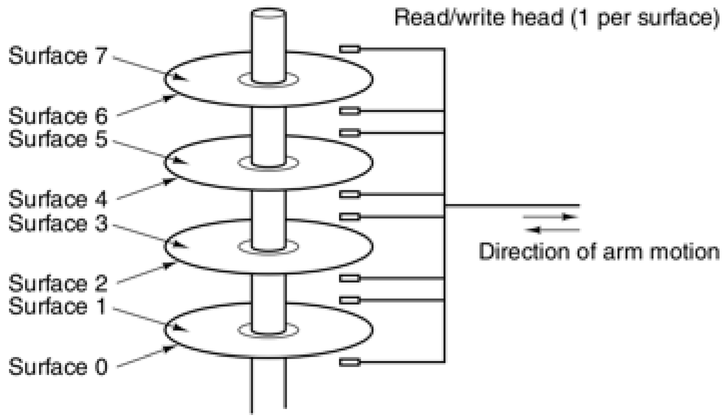
- The programmer is directly manipulating CPU registers and working with specific memory addresses.
- They need to understand how the memory is laid out and how the CPU handles data in registers.
- The programmer is manually loading values from specific memory addresses, performing operations on them, and then storing the result in another memory address.
- Error-Prone: If the programmer accidentally used the wrong memory address, or made a mistake in the ADD or MOV instructions, the program could fail, or worse, overwrite critical parts of memory.

Defining Operating Systems

A virtual Machine Providing Abstractions

- A good abstraction serves two primary functions:

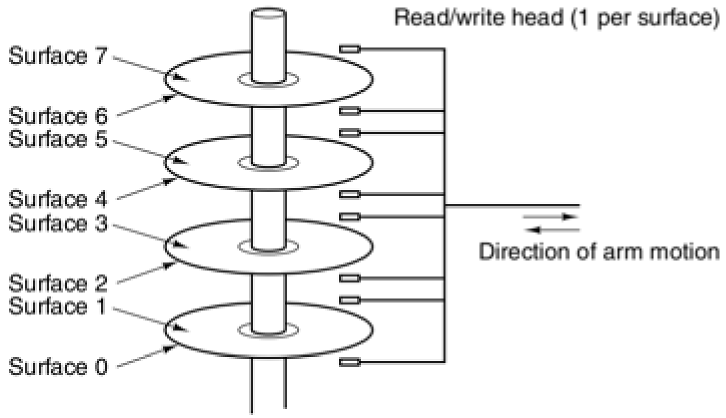
1. Define and Implement Abstractions:
    - This involves creating clear, standardized models of complex hardware or software systems that hide the underlying details. e.g.:
        - **File Systems**: Rather than dealing with bits on a disk, an operating system abstracts this into files and directories.
        - **Processes**: Instead of directly managing the CPU, memory, and devices for each program, the OS abstracts these resources into manageable units called "processes."
        - **Memory Management**: Virtual memory abstracts the complex physical memory layout, allowing programs to think they have access to a continuous block of memory.



Defining Operating Systems

A virtual Machine Providing Abstractions

- A good abstraction serves two primary functions:
  2. Use Abstractions to Solve Problems:
     - Once the abstractions are defined, programmers can use them to focus on solving their specific problems without worrying about the hardware complexity.
     - This allows developers to:
       - Write software more quickly and efficiently.
       - Build complex systems with reusable components.
       - Leverage system resources in a more structured and manageable way.



Defining Operating Systems

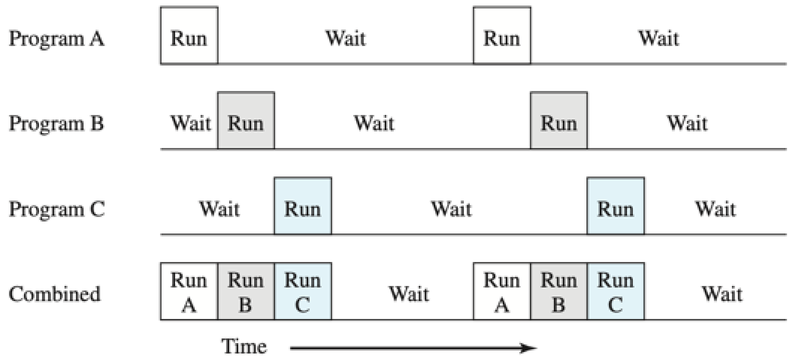
A Resource Manager

- Modern operating systems are designed to handle multiple programs (or processes) running simultaneously, a concept known as multiprogramming or multitasking.
- Goal of multiprogramming:
  - Efficient use of CPU, memory, and I/O resources by interleaving the execution of multiple programs.
- The operating system needs to share system resources between multiple processes.
  - This resource sharing can be categorized into two types: **Time sharing and Space Sharing.**

Defining Operating Systems

A Resource Manager

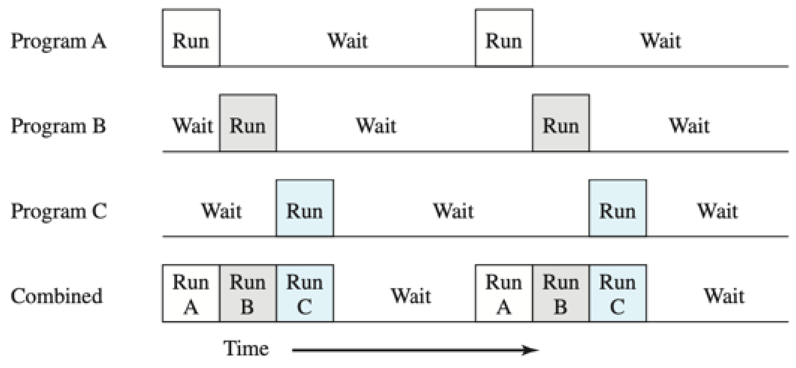
- **Time Sharing:** refers to how resources like the **CPU** or **printers** are shared by multiple processes **over time.**
- **E.g. (CPU Time Sharing or CPU scheduling)**:
  - The CPU can only run one process at a time.
  - The OS uses **time slices** (small intervals of CPU time) to run each process in turn.
  - This is managed by the OS's **scheduler**, which quickly switches between processes so that each process gets CPU time and appears to run simultaneously.
- **E.g. Printers**: if multiple processes want to print, the OS manages a **print queue**, allowing each process to print in turn.



Defining Operating Systems
A Resource Manager

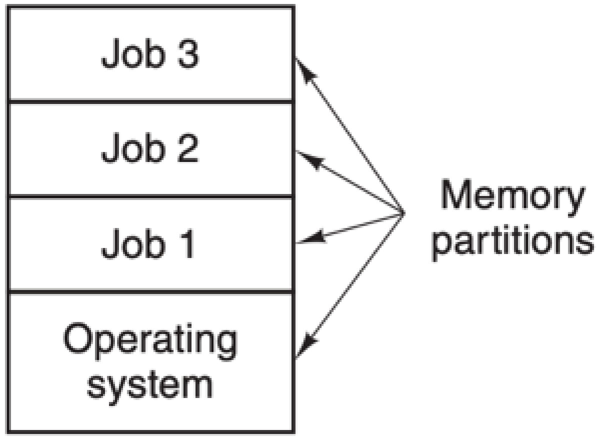
- Interleaving Execution of Processes
  - Context Switching: OS switches between processes, saving and loading their state (process control block).
- Process Scheduling:
  - OS determines the order of execution based on priority or time slices.
    - Examples: Round Robin, Priority Scheduling. (will discuss in detail next week)

- Concurrency Challenges
  - Mutual Exclusion:
    - Ensures that only one process accesses a shared resource at a time.
    - Solved by: Locks, Semaphores, Monitors.
  - Deadlock Avoidance:
    - Prevents processes from waiting indefinitely for each other. (Discuss in detail in Concurrency session)
    - Techniques: Banker's Algorithm, Deadlock Detection & Recovery. (Discuss in detail in Deadlock session)



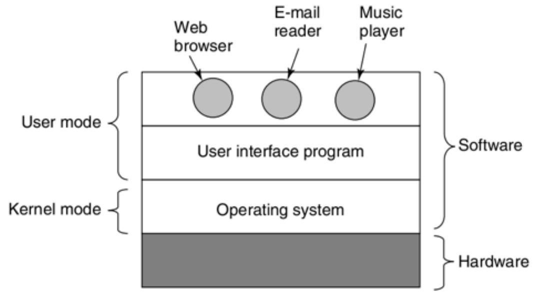
Defining Operating Systems
A Resource Manager

- **Space sharing**
  - Refers to how resources like **memory** and **disks** are shared by processes.
  - The OS must manage how much space each process uses and ensure there is no conflict or overlap between them.
    - OS provides virtual memory to map each process's memory into physical memory locations safely.



Efficient and Restricted Execution

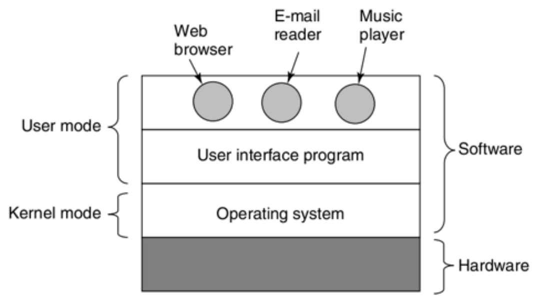
Kernel and User Modes in Modern Operating Systems



- Most computers operate in two modes to ensure security, stability, and control over system resources:
  - Kernel Mode:
    - **Full control over hardware:** The operating system runs with unrestricted access to all system resources.
    - **Can execute privileged instructions:** Direct access to hardware and critical system operations (e.g., managing memory, device drivers).
    - **Handles critical tasks:** Manages process scheduling, memory allocation, and I/O operations.
  - User Mode:
    - **Restricted access:** Applications (user programs) run with limited privileges.
    - **Non-privileged instructions only:** Can only perform basic operations; sensitive hardware and system resources are protected.
    - **Prevents system compromise:** Applications are restricted from performing actions that could harm the system or other processes.

Efficient and Restricted Execution

Kernel and User Modes in Modern Operating Systems

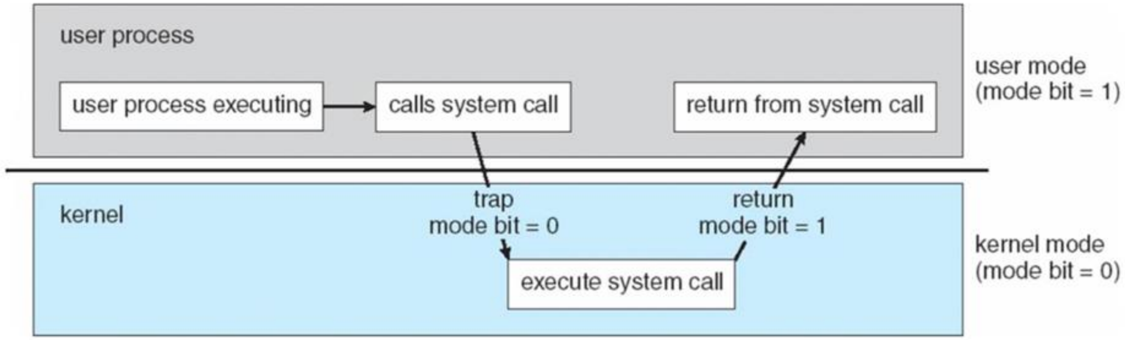


- Privileged Instructions, e.g.
  - I/O Operations: Direct access to devices like disks, printers.
  - Memory Management: Control over system memory allocation.
  - CPU Control: Altering process states, interrupts, or system-wide changes.
- Non-privileged instructions (Subset of Instructions) e.g.
  - Basic arithmetic and logical operations,
  - Instruction that accessing their own memory space,
  - Call functions that do not require hardware access.

Efficient and Restricted Execution

System call and mode switch

- **System Calls**: Applications use **system calls** or **APIs** to request privileged operations from the **Operating System (OS)**.
- **Mode Switching**
  - When an application needs to perform a privileged operation (e.g., accessing hardware), it triggers a **system call**.
  - The **OS switches** from **user mode** to **kernel mode** to execute the operation.
  - Once the operation is complete, the OS switches back to **user mode**, returning control to the application.
- **Example: Disk Read Operation**
  - When an application needs to **read from a disk**, it makes a **system call**.
  - The OS enters **kernel mode** to access the disk, performs the read operation, and then returns to **user mode**.



Example of system calls

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory- and file-system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount, so no umount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

What is Operating System

What can an OS Do for you when you write a Java program?

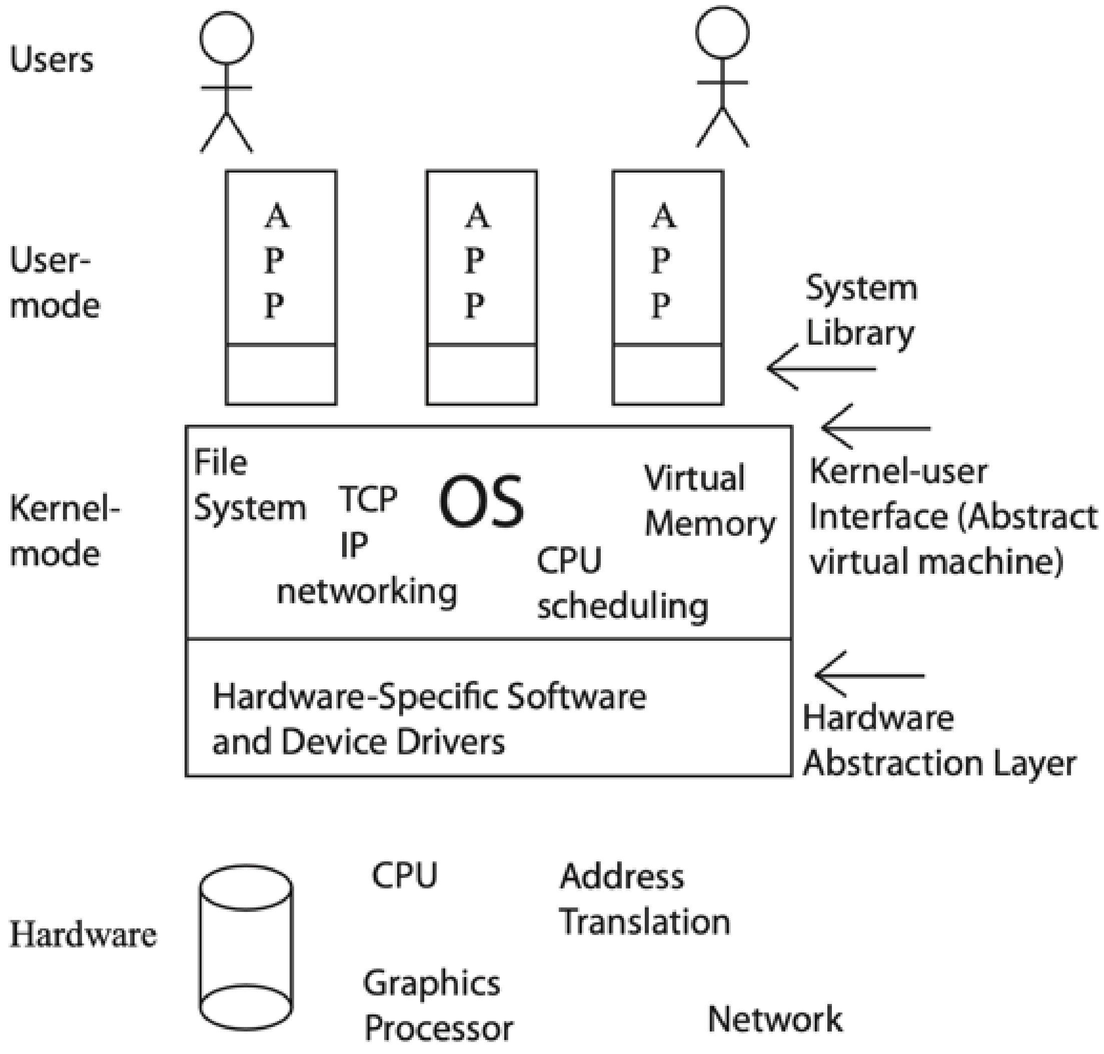
```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class Demo1 {
  public static void main(String[] args) throws IOException {
    FileWriter fw =
      new FileWriter(("C:/Program Files (x86)/test.txt"));
    PrintWriter pw = new PrintWriter(fw);
    pw.close();
  }
}
```

- File Creation: The OS creates the file if it doesn't exist and checks if you have permission to write to the directory.
- I/O Management: The OS manages writing data from the `PrintWriter` to the file, optimizing performance with buffering.
- Resource Allocation: The OS allocates memory for file handling and ensures that resources (like file handles) are properly released when the program finishes.
- Error Handling: The OS checks for issues like insufficient permissions or file path errors, throwing an `IOException` if something goes wrong.
- Security: The OS ensures your program only accesses files it has permission to modify.

Summary

Take-Home Message



- A virtual machine providing abstractions
  - Hide the complicate, messy, low-level hardware interface
  - Files, address, process……
- A resource manager
  - Keep track
  - Grant resource request
  - Account for usage
  - Mediate conflicting request for different programs
- Manage in a controlled manner
  - System Calls
  - User-Kernel Mode