OPERATING SYSTEMS

Course content

1.0 Introduction: Overview of OS functions, architecture, traps and interrupts, kernel;

Reading list

Tanenbaum, A. (2008) *Modern Operating Systems*. *3rd ed.*
Silberschatz et.al. (2005). *Operating System Concepts*. 7th edition
Harris, J. Archer. (2002). *Operating systems*.

Prerequisites

Proficiency in the C programming language.

1.0 Introduction

1.1 Defining an Operating System

An operating system runs on computer hardware and serves as a platform for other software to run on the computer system

I

An operating system is the software component of a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer.

II

An operating system is a program that acts as an intermediary between a user of a computer and the computer hardware.

From the two definitions above, we learn that an OS assumes two roles:

- OS is a ***resource allocator -*** manages all resources; 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.

There is no universally accepted definition of an OS.

1.1.2 Do we need an OS?

- Convenience: ease the use of machine by handling low-level tasks
- Efficiency: maximize throughput by efficient usage of machine resources
- Evolvability: allow development of new features while maintaining compatibility
- Use OS techniques even if you're building controller for microwave oven

1.1.3 Evolution of computing from the OS perspective:

Early computers -- ENIAC to mid 50's:

- Program by switches - no system software
- Devices were very simple - didn't need resource abstraction
- No concurrency (simultaneous multiple processes)
- Application program responsible for all I/O

Simple batch systems - mid 50's - 60's:

- Human OS
- Key-punched card stacks put into 'job queue'- an ordered list of jobs waiting to be processed by a subsystem.
- Step forward: allowed multiple users to share machine
- Step back: no more real-time interaction/interactive debugging

Multiprogrammed batch systems - 60's:

- Took advantage of different time scales at which CPU and peripherals work to improve
  CPU usage
- Introduced multiprogramming: time-MUX processor, space-MUX memory
- Objective: maximize job throughput
- short-term scheduling or "swapping" of jobs

Timesharing systems -- mid 60's - 70's:

- Users work on central machine at interactive consoles
- Computer provides virtual machine (VM) to each user
- Timesharing allows users exclusive control of (comparitively slower) VM
- Objective: provide equitable amounts of processor/memory to each VM

Minicomputers, microcomputers (early PCs) -- late 70's - early 80's:

- Out of the special-purpose room into the office
- Bare bones system software in ROM for device interface
- One user, one program -- huge step back
- But changed the way we view computers (from corporate resource to personal tool)

Mature PC's and workstations -- mid 80's - 90's:

- Evolution to multi-user, multi-process, networked environment

1.2 Computer System Structure

Computer system can be divided into four components:

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

1.3 Functions of Operating system

a) Manages hardware resources of computer

- Time multiplexes the processor (process management)
- Space multiplexes memory (memory management)
- Time multiplexes memory (virtual memory systems)
- Implements file system (file management)
- I/O system (device management)

b) Process management

- Create, destroy, block, run a process
- Allocate/deallocate resources to processes as needed
- Allows multiple users (processes) to share machine

c) Memory management

- Finite amount of memory must be shared between different programs executing concurrently
- Must protect each running program from having its memory area corrupted by others
- Programs often need more memory than is available - how do sort this out? use virtual memory by integrating secondary storage devices

d) File management

- Abstracts operations on low-level storage devices (disks, tapes, etc.)
- Need to do so was instrumental in development of OSs
- Defines organization of file system hierarchy

e) Device management

- Allocate, isolate/share devices (disks, modems, printers, etc.)
- Hides low-level details of control
- Provides generic interface through device drivers

1.3 Operating System Operation

1.3.1 Basic concepts

a) 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 (for I/O for example), OS switches to another job

b) Timesharing (multitasking) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing

- **Response time** should be < 1 second
- Each user has at least one program executing in memory -**process**
- 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

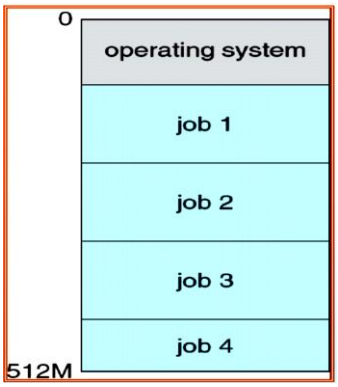


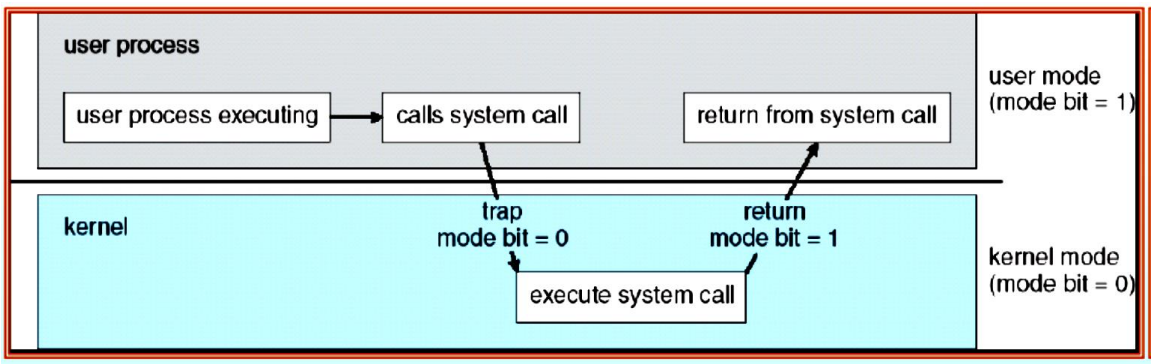
Illustration: Memory Layout for Multiprogrammed System

1.3.2 Operating System Operations

- Interrupt driven by hardware
- Software error or request creates **exception** or **trap -** Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as privileged, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

1.3.3 Transition from User to Kernel Mode

- Timer to prevent infinite loop / process hogging resources
- Set interrupt after specific period
- Operating system decrements counter
- When counter zero generate an interrupt
- Set up before scheduling process to regain control or terminate program that exceeds allotted time



1.3.4 Evolution of kernels:

60's: non-process kernel

- OS kernel is special program
- Had own memory, stack, etc.
- Operated in its own privileged mode

70's: kernel executed within user process

- Common for small minicomputers, PCs
- Typified by UNIX
- OS services accessed via normal subroutine calls
- Require scheme to enter SUP to give OS access to resources (and return to USER mode when done)

80's onwards: Microkernels

- Minimal fast kernel that provides only basic services
- Non-kernel modules communicate largely via message-passing

1.5 Operating System Structure

1.5.1 Operating System Services

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), Graphics User Interface (GUI)
- Program execution -The system must be able to load a program into memory and to run that program, 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.
- Communications –Processes may exchange information, on the same computer or between computers over a network; Communications may be via shared memory or through message passing (packets moved by the OS)
  - Error detection –OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing:

- **Resource allocation -**When multiple users or multiple jobs running concurrently, resources must be allocated to each of them; Many types of resources -Some (such as

CPU cycles, main memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code.

- **Accounting -**To keep track of which users use how much and what kinds of computer resources
  - **Protection and security -**The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **Protection** involves ensuring that all access to system resources is controlled
      - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
    - If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

1.5.2 System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based
- systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

1.5.3 Operating System Design and Implementation

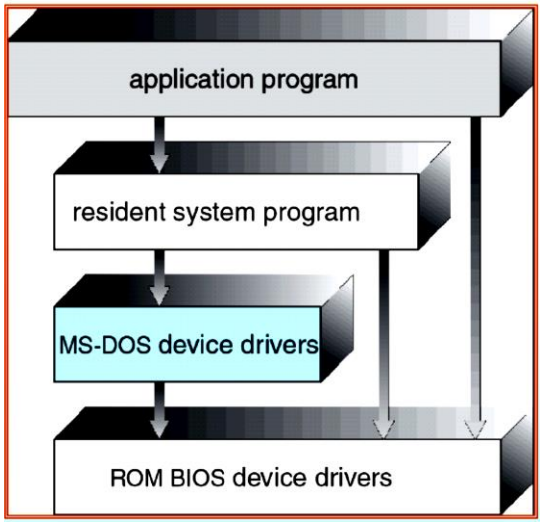
- Design and Implementation of OS not "solvable", but some approaches have proven Successful.
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
- *User* goals and *System* goals:
  - *User goals* –operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - *System goals* –operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.
- Important principle to separate
  - **Policy:** What will be done?
  - **Mechanism:** How to do it?
- Mechanisms determine how to do something, policies decide what will be done
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later.
- Two main ways of structuring OS **– *simple* and *layered approach***

a) Simple Structure

- OS developed as simple, small, having limited functionality and then with time they started to grew up beyond their scope, e.g. MS-DOS
- Not very much efficient

Illustration: MS-DOS

- Written to provide the most functionality in the least space
- Not divided into modules
- Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.

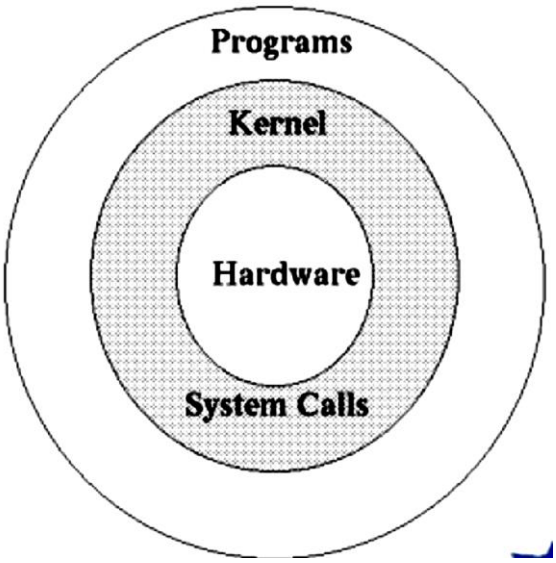


Drawbacks

- Application programs are able to access the basic I/O routines -means they can directly read or write to the basic display and disk drives - vulnerable to the malicious programs, causing entire system crashes
- Limited to hardware of its era

b) Layered approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



Drawbacks:

- Problem in defining the functions of every layer because a layer can only use its lower level layers for its functioning.
- tend to be less efficient than other types; for instance consider an example, when an user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which then call the memory management layer, which in turn call the CPU scheduling layer and then passes to the hardware. In this whole process, the each layer add its corresponding overhead result in that system call takes more time in comparison to the non-layered approach