Fundamentals of Operating Systems

Definition

- No universally accepted definition
- The OS is software – it is a program which “virtualises” your computer
- A program, usually written in C or C++
- “Everything a vendor ships when you order an operating system” is good approximation.
- A software layer between the hardware and the application programs/users
  - provides a virtual machine interface
    - easy to use (hides complexity)
    - safe (prevents and handles errors)
  - resource manager
    - allows programs/users to share the hardware resources in a protected way
    - fair and efficient

Objectives

- Loads and runs other programs
- The OS provides a consistent way to for application proggram to work on whatever hardware you have
- Resource allocator
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- Control program
  - Controls execution of programs to prevent errors and improper use of the computer
- To hide details of hardware by creating abstraction
- To provide a pleasant and effective user interface

Goals

- User goals
  - convenient to use
  - easy to learn
  - reliable
  - safe
  - fast
- System goals
  - easy to design
    - implement
    - maintain
  - efficient
  - error-free
  - reliable
  - flexible

Computer System Structure / Key Concepts

Hardware
- CPU
- memory
- I/O devices

Application programs
- define the ways in which the system resources are used to solve the computing problems of the users

Operating system
- Controls and coordinates use of hardware among various applications and users

Users
- People
- Machines
- Other computers

Kernel
- The one program running at all times on the computer

Bootstrap
- program is loaded at power up or reboot
- Typically stored in ROM or EEPROM, generally known as firmware
- Initialises all aspects of system
- Loads operating system kernel and starts execution
- The OS starts executing the first process e.g. init and waits for some event to occur.

I/O Devices
- 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
- 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

Interrupts
- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt
- A trap is a software-generated interrupt caused either by an error or a user request
- An operating system is interrupt driven
- causes the processor to save its state of execution, and begin execution of an interrupt handler.
- commonly used technique for computer multitasking, especially in real-time computing

Virtual Machine Abstraction

OSs provide a virtual machine that enable us to use the hardware in a more convenient and efficient manner

Reasons for abstraction

- the code to control peripheral devices is not standardised. OS provide subroutines called device drivers to perform operations on behalf of programs e.g. I/O operations
- the OS introduces new functions as it abstracts the hardware. e.g. the file abstraction so that programs do not deal with disks
- the OS transforms the computer hardware into multiple virtual computers, each belonging to a different program. Each program that is running views the hardware through the lens of abstraction.
- the OS can enforce security through abstraction

Types of Os's

Batch Processing

Simple Batch Systems

- User jobs are submitted in sequential batches on cards or tape
- No interactions with the user
- Jobs are executed one at a time

Multiprogrammed Batch Systems

- Multiprogramming/Multitasking prevents processor time being wasted while waiting for I/O
- requires some form of memory management and scheduling algorithms

Time-sharing

- Provides a computational service to many online users concurrently
- The OS interleaves the execution of each user program in a short burst of computation
- Interactions allowed
- Sharing CPU time and other resources
- Protection
    - With multiple jobs in memory they must be prevented from modifying each others' data
    - Ensure file system allows access only to the authorised users of each file

Real-time

services on-line external processes having strict timing constraints on response.
If the processes are not handled promptly with a critical period of time then the process is seriously degraded or misrepresented

- Network
  - existence of multiple computers
  - can log into remote machines
  - copy files from one machine to another
  - Each machine runs its own local OS
  - Can support a diskless workstation environment (LAN)
- Distributed
  - Users are not aware of the existence of multiple computers
  - Treat an ENTIRE set of machines as a ‘single’ system
  - Complex
- History
  - 40s: only hardware, no OS
    - programs were entered one bit at time on rows of mechanical switches (plug boards).
    - ENIAC: 1945 - 1955
  - 50s: first OS by General Motors
    - Encode program as punched cards
  - 60s-70s: mainframes with professional operators
    - Instructions as punched cards
    - Timesharing/Multiprogramming and multi-user using terminals
  - 80s –present:
    - personal computers
      - GUI-based OSs
      - Command processors
      - ++es change
    - OSs closely tied to the architecture of the computers on which they run.
      - In the beginning
        - Expensive Hardware
        - Cheap People
        - Goal: maximize hardware utilization
      - Present day
        - Cheap Hardware
        - Expensive People
        - Goal: make it easy for people to use computers

OS Services

- User interface
  - Command-Line, Graphics User Interface, Batch
- Program execution
  - The system must be able to load a program into memory, run it, end execution, either normally or abnormally (indicating error)
  - A running program may require I/O, which may involve a file or I/O operations
    - File-system manipulation
      - read/write files and directories, create/delete, search, 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 errors that 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
- efficient operation
  - Resource allocation
  - Accounting
  - Protection and security

Dual-Mode of Operation

- User mode

- Kernel mode
- System Calls
  - Typically written in a high-level language (C or C++)
  - Accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
    - Program portability
    - Actual system calls may be complicated and difficult to work with.
  - Types of System calls
    - Process control
    - File management
    - Device management
    - Information maintenance
    - Communications

System programs

- System programs
  - Provide a convenient environment for program development and execution.
  - They can be thought of as bundles of useful system calls.
  - Most users' view of the operating system is defined by system programs, not the actual system calls

Mode bit

- provided by hardware: kernel(0) or user(1)
- Provides ability to distinguish when system is running user code or kernel code
- Instructions designated as privileged, only executable in kernel mode
- System call changes mode to kernel, return from call resets it to user