MIT World Peace University

Computer Networks
Second Year B.Tech Semister 3
Academic Year 2022-23

Operating Systems

Notes from Tananbaum and Classes

Prepared By

P34. Krishnaraj Thadesar

Batch A2

August 30, 2022

Contents

1 Introduction

A modern computer system consists of one of more processors, RAM, ROM, IO devices, and a lot of stuff, this is like common knowledge, and known to everyone. Which means we know that a computer is alot of things happening at the same time, and if the programmer was to know all the things that were happening in the computer, and he or she had to deal with whether or not the frequency of pulse sent to the motor of the hard disk is greater than the required amount every time they needed to create a variable or something, then it simply would not be possible to write any code.

2 Operating System

What is an operating system?

1. It is a program that acts as an intermediary between a user and the hardware.
2. It is also the program taht controls the execution of application programs.
3. It allocates resources effectively.

The main Objective is convenience, efficiency and providing an environment. It has a kernel, and is always running all the time.

Modes of Operation The Operating system is the most important piece of software and runs in the kernal mode. The rest of the softwares run in the user modes.

Kernal Mode This is where You get complete access to all the hardware and software components of the system. It is also called the supervisor mode. You can do anything here and that is why it needs to be kept secure, and is often restricted to the users.

User Mode This is where all the interaction takes place from the user. Most of the software that we see is written to work only in this space.

The User Interface Program This is the shell or the GUI. As the name suggests, it is the way to interact with the user and the system.

It is not important that every kind of operating system has these modes. Embedded systems for examples may have only 1 mode, coz they are often simple in design.

- Operating systems are much bigger than normal software, something like 5 million lines.
- They are mostly in the kernel mode, but the boundary often blurs.
- Coz they are so long to write, companies would often not mess with what they already have, and just build on that than starting from scratch.

It is an extended machine The hardware is difficult to program, and OSs use abstraction very nicely to make us programmers feel that it is easy. Abstraction is therefore a key to managing any kind of complexity. A good abstraction can turn a nearly impossible task into 2 manageable ones. To define and implement them, and to use them to solve the problme.
It is the job of the OS to turn some ugly interface into a beautiful one.

It is A Resource Manager This is again self explanatory, but more importantly, it includes multiplexing resources in time as well as in space. The usual example is the one where 3 programs might want to print something on the screen, and save some data on the disk, if the OS doesnt bring order to chaos, its just gonna be all together and messed up.

3 History of the OS

Im pretty sure this isnt needed for the exam, but id rahter write it coz its interesting.
OS types have been closely related to the type of CPU architecture.

3.1 Babbage - 1850s

The father of the computer. This guy was a mathematician. He wanted to make an analygical engine, and obviously that means everything was all mechanical, which meant there wasnt much software. But as he realized he couldnt coordinate everything without a software, he hired a young woman named Ada Lovelace, who was the daughter of Lord Byron, that poet dude. The programming language Ada is therefore named after her.

3.2 First Generation (1945-1955) Vacuum Tubes

That date should ring some bells, no wonder, the advancements came at this time. The first functioning digital computer was built at IOwa state uni with 300 vacuum Tubes. All these took seconds to perform even simple calculations. All programming at this time was done by pure machine language, or by literally connecting thousands of cables.
By the 50s, the job of the operating system had to be done by programmers. You had to manually write your program in a punch card, and write your name and purpose on the signup sheet, and once you did that you would wait for your turn and insert your program and wait for the results.

3.3 The Second Generation (1955-65) - Transistors

With transistors, we could get rid of vacuum tubes, and therefore increase reliabilty, while reducing size. Now these machines are called mainframes. You still have to do essentially the same thing, excpet now you write the program in Assembly or FORTRAN, and punch it to cards and wait for output.

To make this better, people started using blocks, and a different computer for loading programs into tapes, so things were faster. The output was printed by a computer that wasnt connected to the main line, and was therefore *Off Line*

The page that the programmer had to submit however had much the same structure as we see now. They first had to write the JOB card, specify run time, then a FORTRAN card, telling the OS (usually IBM SYS OS) to load the FORTRAN compiler from the system tape, then the program to be compiled, then a LOAD Card and finally the RUN and END card. In this way an entire program was submitted to the system.

3.4 The Third Generation - ICs and Multiprogramming

The development of ICs meant that you could now fit larger computers in a smaller size. IBM Tried to integrate different types of computers, into a series of computers that would all be compatible with each other, the IBM 360 Series. It could handle commercial as well as scientific computing. There was a family of such computers meant for different tasks, but all were meant to be compatible with

each other. Now we know what that means. The Operating system was Huge. It had to manage running efficiently on a potato computer and a solid scientific computer. This meant thousands of computers were needed, and every release of this OS was riddled with bugs. And yet it was still a success.

- It introduced multiprogramming. This meant you could hae serveral jobs run at the same time. Explained more later. So now a lot of the IO time was reduced, and CPU was suddenly very efficient.
- It could read jobs from cards onto the disks as soon as they were brought into the computer room. This was called SPOOLing
- It introduced timesharing, as programmers didnt wanna wait for so long to debug their programs.

3.4.1 MULTICS

General Electric, MIT and Bell Labs envisioned that like the electricity grid, you could have something like a computer grid, where you could have a bunch of computing power for everyone in Boston. You should just be able to connect and use it. It was kinda popular and saw mixed success. It was later bought by Honeywell. As you might notice this is kinda like the server design.
Another major development during the third gen was the growth of minicomputers. Ken Thompson, found a small minicomputer that was a stripped down one user version of MULTICs. This was later developed into the UNIX Operating System that we all know and love.

UNIX then branched out into System V and BSD. To make it possible to write programs on any of these distributions, IEEE Developed POSIX. POSIX is a minimal standard system call interface for UNIX systems. And the BASH shell is based on it.
This then led to the development of MINIX for educational purposes, and later, the desire for a free production as opposed to just educational version of MINIX is what led a Finnish Student, Linux Torvalds to develop Linux.

3.5 Fourth Gen of Computers. (1980- Present)

4 Shells and Scripts

System Calls System calls are what are called by the operating system to the kernel. When you say things like printf and all they then invoke system calls Examples are read(), write() etc

4.1 Shell

The shell is simply another program on top of the kernel which provides a basic human os interface.

There are different types of shells

1. /bin/csh - It is the C Shell
2. /bin/tcsh - Enhanced C Shell
3. /bin/sh - The Bourne Shell / POSIX shell
4. /bin/ksh - Korn Shell

5. /bin/bash - Korn Shell Clone from GNU

All linux systems use the bash shell as the default.

4.2 Scripts

A script is a bunch of lines of the script written in a plain text file.

Why should we write shell script?

1. Shell script can take input from the user or file and then output them on the screen.
2. Useful to create our own commands. Save lots of time
3. To automate some tast of daily life.
4. System administration part can be automated.

Practical Examples where you can use scripts

1. Monitor your systemdata backupFind out what process are taking up resources
2. Find out which memory is free
3. Find which users are logged in
4. Find out if all necdessary network services are running etc.

5 OS Components and Functions

5.1 Process Management and CPU scheduling

A Process is when a part of your program is in its execution state.
A program is on a higher level than the process. One Program can have a lot of processes. On an even higher level is the job or the task. The tast is the highest level. Whenever you say something exclusively in your program, then doing that is caleld a process.
A process needs certain resources, including CPU time, memory, files, and IO devices.

The Operating system is responsible for the following activities

1. Process creation and deletion
2. Process suspension and resumption
3. Provision of mechanisms for process synchronization and Process communication

5.2 Memory Management

1. Memory is a large array of words or bytes. Each with its own address
2. It is a repository of quickly accessible data shared by the CPU and IO devices
3. Main memory is a volatile storage device. It loses its contents in case of a system failure.
4. The Operating system is responsible for the following activities in connections with memory management
   (a) Keeping track of which parts of the memory are currently being used and by whom.
   (b) Deciding which process to load when memory space becomes available
   (c) Allocating and deallocating memory space as needed.

5.3 File Management

There are 2 types of Files. Sequential and Direct Access Files.

1. A File is a collection of related information defined by its created. Commonly files represent programs and data
2. The operating system is responsible for the following activities in connections with file mangement.
   (a) File creating and deletion
   (b) Directory creation and deletion
   (c) Support of primitives for manipulating files and directories.
   (d) Mapping files onto secondary storage. File backup on stable or non volatile storage media.

5.4 IO System management

1. Control of devices connected to computer
2. IO Devices vary widely in their function and speed, so different methods are needed to control them.
3. Device drivers are required to provide an interface to IO devices
4. Also the IO system consists uses buffering to take care of speed difference between IO devices and processor.

6 Types of Programming

1. Uni Programming
   (a) Process must wait for IO instruction to complete before proceeding
   (b) The processor spends a certain amount of time executing, until it reaches an IO instruction. It must wait until that IO instruction concludes before proceeding.
   (c) 2 Programs cannot be run at the same time. The OS takes care of this, and makes sure that that doesnt happen.

   (d) This is usually run in Single core CPUs.

2. Multi Programming

   (a) Several jobs are kept in the main memory at the same time and the cpu is multiplexed among them.

   (b) There must be enough memory hold hte OS and one user program.

   (c) When one job needs to wait for IO, the processor can switch to the other job, which is likey not waiting for IO.

   (d) So you might confuse this with multithreading, its not that, we are still running stuff serially on a single cored CPU, but it feels like a lot of things are happening at the same time.

   (e) So you can actaully simulate doing many tasks at the same time, while still performing them serially. The CPU does not need to wait, if the processes arent output dependent on each other.

   (f) The CPU is *multiplexed* here

7 Types of Operating Systems

7.1 Time Sharing Systems

1. Can be used to handle multiple interactive jobs.
2. Processor time is shared among multiple users.
3. Multiple users simultaneously access the system through terminals, with theos interleaving the execution of each use program in short burst or quentum of computation.
4. An operating system that uses multi tasking like seen above, then that would also be a time sharing system.

7.2 Distributed OS

1. Distributed system is a collection of loosely coupled (less dependent) processors interconnected by a communications network.
2. Processors variously called nodes, computers, machines, hosts.
3. gves the impression thatthere is a sigle operating system controlling the network.
4. Users not aware of multiplicity of machines Access to remote resources similar to access to local ones.
5. Advantages are Resource sharing, Reliability, Communication, Computational Speed up.
6. An Example is banking and stuff.

7.3 Real Time and Embedded OS

1. Rigid time requirements are placed on operation of a processor or flow of data
2. Often used as a control device in a dedicatd application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems.
3. Well defined fixed time constraints and rocessing must be done within define constraints.
4. Real Time Systems may be either hard or soft real time.
5. Example is a Washing Machine for Embedded OS, the OS in Heart Monitor Systems would be a Real Time system.
6. Real time and Embedded OS go hand in hand.

8 Operating System Structure

8.1 Types

8.1.1 Monolithic

The Main point here is that everything is in the kernel. This makes it rather efficient, but also a little insecure.

1. Every component contained in kernal.
2. Direct communication among all elements.
3. Highly Efficient
4. Problems are complexity, new devices, emerging tech enabling, protection etc.

In the kernal Space, we have several things in the kernal that can be called by the system call interface. Those things are:

1. Memory Manager
2. Processor Scheduler
3. Interprocess Communication
4. file system
5. Input/Output
6. Network Manager etc.

8.1.2 Layered

1. The Problem with monolithic is that it was kinda less secure.
2. Flow of communication is different.
3. Its like a hierarchical thing, where you need permissions and stuff, so this makes it more secure.
4. The different layers would be the computer hardware -> operating systems -> utilities -> Application Programs.

8.1.3 MicroKernel Approach

- Kernel is modularized using micro kernel approach.
- It removes all non essential components from kernel and implements them as system and user level programs.
- It provides minimal process and memory menagement, in addition to communication facility.
- Other OS servies are provided by processes, called as servers that run in user mode and are treated like any other application by microkernel. (eg. diff file organizations can be implemented as one service), Device drivers, file systems, security services.
- It provides communication between client program and various services that are running in user space.
- communication is provided by message passing. called MPI.
- Clients and services communicate by exchanging messages with microkernel and not directly.
- Things like application programs, file systems device drivers etc all come in the user mode here. In the kernel mode, you have interprocess comm, memory management, and CPU Scheduling.
- Notice the differences here, user can access a lot of things here as opposed to monolithic kernels.

Benefits of Microkernel Design:

1. Uniform Interface: Uniform interface on requests made by process. Processes are not distinguished as user or kernel level, since services are provided by means of messages passing between them.
2. Extendibility: Easy of extending OS
3. Flexibility: New services are added to the user space and no modification required at kernel level. Existing features can be subtracted to produce smaller and efficient implementation.
4. Kernel level modifications are very few.
5. Portability OS is easier to port from one Hardware design to another.
6. Provides more security and reliability., since most services are running at user level.
7. If service fails, rest of OS remains unchanged. This is so that no one can interfere with Kernel architecture and stuff.
8. Examples would be Tru64Unix, or Apple MacOS