Skip Debugging, Generation in text book.

**Operating System Services**

-UI: Command Line, GUI, Batch. Our way to interact with system  
--GUI: What we see on screen display  
--Batch: Process that would run at certain times  
--CMD: Interact with OS through CMD. Allows direct command entry

-Program Execution:

-IO Operations:

-File system manipulation requires a file to not be edited by multiple processes at once.

-Communications

-Error Detection

-Resource Allocation: Allocate resources on process start

-Accounting: TO make sure we keep track of time allocated to CPU for certain process. For stopping processes

-Protection and security  
--Protection: Protect processes from one another  
--Security: Protect system and processes from malicious attacks

System calls, below UI, and above all else (eg IO operations): This allows the interaction between the 2

**System Call**

**-**Allows us to call a function provided by the OS

Eg .close(), open()

No Call:  
ofstream – ie file stream

The program/process, is a different process than the system call.  
ie Your code process, and a write to file process (system call)

**Current working directory:** directory an application normally allocates when accessing a process (**NBNB Terminology)**

**Remember, applications cant create files, only the OS can.**

**System Call Parameter Passing (NB)**

Three general methods to pass parameters to the OS

1) Pass parameters in registers  
-But what if you have more info, than registers

2) Have a pointer pointing to a block/table (in memory) storing all the parameters

3) By using stack memory, when calling a function, you push parameters of function on stack, and pop parameters from stack (Most used by OSes these days)

**Trap (terminology):** When you flush data to the system call (?)

**NBNB must know all types system calls (textbook 2.4)**-what type of instructions need a system call  
-ie spot lines of code that generate system call

The location of Windows system calls is unknown.

**System Programs**

File Management: file explorer

Status Information: Task Manager

File Modification: Notepad

Programming-Language Support: Java-script (?)

Program loading and execution: .Net Framework

Communications: pipe, rdp, vnc, secure shell

Background services: Graphics, UPP host services (universal plug and play), SQL server (starts before login)

Application programs: Chrome

**Implementation of OS**

Create a C compiler, as easier than assembly.

High level languages its easier t port to other hardware.  
-but slower

Emulation can allow non-native hardware to be run

**OS Structure**

MS-Dos was a simple structure. Written to provide the most functionality In the least space.

Application can send instructions to printer drivers, instead of bios, to print.

**Traditional Unix System Structure**

Monolithic kernels. Whenever you get a new file system, you have to recompile OS.

Most modern ones are modular, = non-monolithic kernals.

**Layered Approach**

Each layer can communicate/call 1 layer below it.

Advantage: simpler.  
-Debugging and verificantion become easier

Disadvantage: It lates a lot longer for something to flow down from layer n to 0

**Microkernel system structure**

User Mode: eg File system, drivers, applications

Kernal Mode: Cpu Scheduling (Decides which process will own the CPU use while another process is suspended. Also makes sure when process suspended, next process in line is available and ready for use), memory management, interprocess communication

Hardware

Communication can occur between User <-> Kernal <-> Hardware.  
Using basic input, output handler (message passing)

Goal: Create strong, small pieces of code to run in microkernel, so as to minimise bugs and complexity. (More reliable)

Easier to extend a microkernel  
Easier to port OS to new architecture  
More secure

Detriment: Perforance overhad of user space to kernal space communication  
-eg Retireiveing a file  
Application -> Kernal -> file system -> kernal -> application

**Modules**

Loadable kernal Modules: Can load eg File system, Only when needed.

While Android is free, google processes, such as Play Store, is not free.

**System Boot**

ROM: Holds firmware used for initial boot 9Bootstrap loader)

Bootstrap Loader locates kernal, loads it into memory, and start it.x