1-1: A system is the “whole” that results when a number of things have been grouped together In a specific manner for a particular purpose. 3 Requirements:

1. A set of 2 or more elements
2. Interconnections/Interactions between elements
3. Interacts with elements in its environment

1-2: Open systems interact with their environment, Closed do not.

1-3: Common examples of systems: File Systems, Operating Systems, Audio Systems

1-4: System of systems have one or more systems and element and elements. Each system is referred to as a system. Systems need to interact in some way.

1-5: A computer system is a system of systems. It blends hardware, firmware and software. It is a collection of subsystems like the OS and file system.

1-6: Definitions are as follows:

System: Defined above

Subsystem: Defined above

CPU: Computer Processing Unit, where register memory, the control unit, and the ALU is stashed

ALU: Arithmetic Logic Unit, piece of the CPU that does the heavy lifting for mathematical operations.

RAM: Random Access Memory, where we store temporary information that will be deleted on shutdown.

ROM: Read Only Memory, stored in the BIOS chip of a motherboard to be read and not modified.

DBMS: Database Management System, a way of defining and enabling operations for a database.

File System: How our information is stored with respect to itself.

NIC: Network Interface Card, piece of hardware that enables us to connect to a network.

TCP: Transmission Control Protocol, A guaranteed delivery method to deliver memory over a network

IP: A unique series of numbers that identifies a server on a network.

2-1: Definitions:

Single User: 1 user can use the OS at one time.

Multi-User: More than 1 user can use the OS at one time.

Single-tasking: A task needs to finish before the next can begin

Multi-tasking: Multiple tasks running at the same time.

2-2: Definitions:

Batch Processing: One job at a time, jobs do not get swapped out or interrupted, typically non-interactive, jobs own all resources. Easy to develop and administer

Multi-tasking: OS constantly switching between processes permitting them to use the CPU. Multiple jobs at the same time, interactive. Complex to develop and administer.

Non-preemptive Context Switching: Processes voluntarily yield the CPU via system calls. A process will run until it blocks.

Preemptive Context Switching: More common, OS forces a switch to another process.

2-3: The OS and CPU work together to manage “virtual memory”. Memory identified on the actual CPU is abstracted into “virtual memory locations”. The OS + CPU take care of getting memory where it actually will be sent or saved. Memory is organized into 4KB to 8KB “pages” that are loaded by the OS into RAM on demand from disk.

2-4: Recall the following commands:

mkdir: make a directory

touch [fileName]: create a file

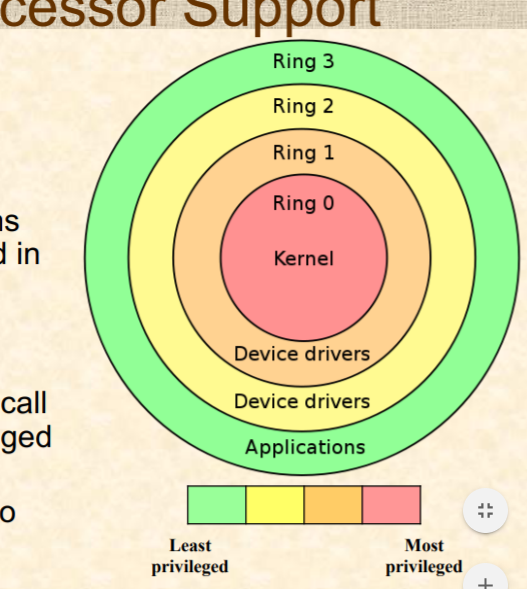
Mv [fileName] [location]: move a file

cp [fileName] [location]: copy a file

cd [location]: move into a directory

2-5: OS provides an interface in the form of device drivers to enable interaction between an OS and peripherals.

2-6: We restrict instruction calls only within a certain ring and above. It takes special instruction calls to move down in rings



Each user on an operating system has a unique ID called a uid. Linux enables the user to assume the role of another uniqueid using sudo, called the effective user id. Linux also enables users to be placed into groups, each of which have access to certain resources and abilities.

3-1: System calls run in privileged kernel space, they are designed to be quick, and they call an OS via it's API. These are implemented in Assembly.

3-2:

1. Implementation in Assembly
2. High Level language wrappers to streamline it
3. Binary executable program

3-3: strace: shows us the path from process beginning at exec until we hit exit\_group

4-1: Influenced by

1. Hardware Configuration(Plug + Play)
2. Nature of processing(Batch or Interactive)
3. Application requirements(Availability, Fault Tolerance, Security)

4-2 + 4-3: Definitions:

Monolithic: Whole thing is one giant clump with everything together. (One program)

Adv: Simple + Easy to dev, small and compact

Dis: Does not scale, hard to maintain.

Microkernels: Core is a well defined kernel and calls are made and passed through that kernel.

Adv: Easily port OS software

Dis: Lots of calls can slow overall machine, performance impact on I/O operations

Layered: Layer1 passes data to Layer2 passes data to Layer3….

Adv: Replacement of layers is easy, easy to maintain, independent layer dev

Dis: More API’s to deal with, tunneling takes a long time

Hypervisor: Uses hardware emulation in the form of virtual machines

Type-1 Hypervisors: All processes are done without an OS

Type-2 Hypervisor: All processes are done inside an OS

Adv: Easy to develop multi-tasking software

Dis: Can slow overall machine

Client-Server: An individual sends data over a network

Adv: Distributed Computing, effective load balancing, better fault tolerance

Dis: System calls are expensive, managing distributed clients is harder.

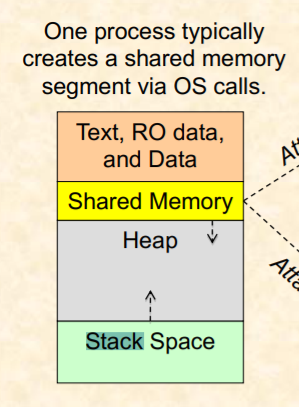
5-1: Booting is how we load an Operating Systems

5-2: BIOS starts up -> BIOS checks system inventory -> BIOS loads OS’s MBR(Master Boot Record) -> First 512 bytes of OS are loaded into memory -> BIOS runs program in MBR

5-3 OS Loaders/Boot Loaders are always present in the MBR. It is just another piece of software that loads the actual OS. A 1 stage loader directly loads the OS into memory. A 2 stage loader loads another loader that loads the OS into memory. We will use stage 2 loaders if 512 bytes is too small for the loader.

6-1 A process is a running program in memory. It is a program with a state and information.

6-2 + 6-4



6-3: ps and top can both be used to view processes running, top displays continously, ps is a snapshot.

6-5: kill command sends a request to a process to terminate itself

6-6: New->Ready->Running->Waiting->Terminating

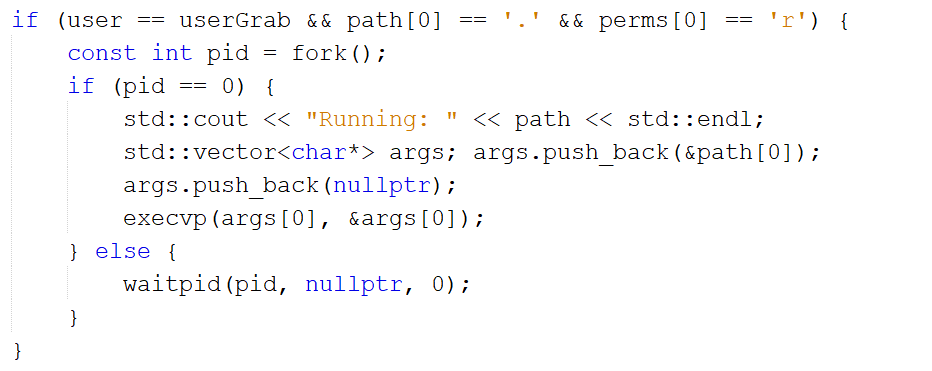
Running-->Ready: When a program running on the CPU is interrupted during processing, it will go from running to ready state.

Running → Blocked: If the CPU is waiting on Input or Output to process or for an additional event to occur, CPU will swap from running to blocked

Blocked → Ready: When a required Input or Output is finished processing, or a required event finishes, CPU will swap from blocked to ready.

Ready → Running: At the discretion of the CPU scheduler, it will dispatch processes from ready to running as designed.

7-1-3:



Swap nullptr to a reference to an int.

8-1: IPC(Inter process communication) is used to build software piplelines of connected processes, information sharing, seepdup computation, increase loosely coupled modularity, and make systems more fault tolerant.

8-2: Shared Memory- A shared region of memory is designated via the OS. Processes can exchange info using that location utilizing pipes or FIFOs. Message passing - Processes send messages to each other, which may go over network or use a special layer of the OS.

8-3: Two types of pipes:

1. Anonymous/Ordinary pipes: Exist only when processes are running and are unidirectional.
2. Named Pipes or FIFOs: Independent of communicating processes, unidirectional in Linux, bidirectional in Windows.

8-4: mkfifo telephone makes a FIFO called telephone

sudo a+rw telephone gives read/write perms to it

8-5: Anonymous pipes ps -fe | grep dunnnm2

9-1: File systems help us organize our data in a cohesive, streamlined pattern.

9-2: relative/a/b/foo.html /home/dunnnm2/leadingslash/getit.html

10-1-3: Addressed above

10-4: g++ -o execName project.cpp

./execName

10-5: pid -> identifier of current process

Ppid -> identifer of parent process

ps shows pid and ppid

10-6: & runs stuff in the background

fg brings background to the foreground

11: Common syntax, other than

method(std::string str) pass by value

method(std::string& str) pass by reference

method(std::string\* str) pass by pointer

12: Mostly self explanatory other than use string streams with >> int x to convert string to int

13-2: int[10] x;

Int[10][10]y;

13-4: argc - 1 = actual number of arguments

Argv[0] is the name of the executed program

Argv[1] is the actual first argument

14: Rao doesn’t care about pointers rn

15-1: std::vector<string> stringVec;

vec.push\_back(“item1”);

15-2: Vectors are arrays with ArrayList equivalent operations

15-3: Shown above

15-4: Same as above

15-5: using namespace std: removes neccessity of scoping character

using StrVec = std::vector<std::string>;

15-6: size() - get size of vec, [0] grab 0th element, erase(“bad”) removes an element, clear() - clears all elements

15-7-8 trivial

16-1: std::unordered\_map<std::string, int> x;

16-2-7: Check previous lab code

17-1: int x >> cin; cout << int x

17-2-5: Check previous lab code

18: General questions

19: Self explanatory, just remember scp [FROM] [TO]

(Note by Steven): If you want scp a folder use scp -r [FROM] [TO]