Study Guide Chapter 1 to 3

**Ch.1 - Introduction**

* An OS isa program that acts as an intermediary between a user of a computer and the computer hardware
* Goals: Execute user programs, make the comp. system easy to use, utilize hardware efficiently
* Computer system: Hardware ↔ OS ↔ Applications ↔ Users (↔ = 'uses')
* OS is:
  + Resource allocator: decides between conflicting requests for efficient and fair resource use: CPU time, memory space, file-storage space, I/O devices, and so on
  + Control program: controls execution of programs to prevent errors and improper use of computer
* Kernel: the one program running at all times on the computer
* Dual-mode operation allows OS to protect itself and other system components – User mode and kernel mode
  + Some instructions are only executable in kernel mode, these are privileged
  + When the computer system is executing on behalf of a user application, the system is in user mode.
  + When a user application requests a service from the operating system (via a system call), it must transition from user to kernel mode to fulfill the request.
* Bootstrap program: loaded at power-up or reboot. Initializes all aspects of the system, from CPU registers to device controllers to memory contents
  + Stored in ROM or EEPROM (known as firmware), Initializes all aspects of system, loads OS kernel and starts execution
* I/O and CPU can execute concurrently
* Device controllers inform CPU that it is finished w/ operation by causing an interrupt
  + Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which is a table of pointers to specific interrupt-handling routines
  + Incoming interrupts are disabled while another interrupt is being processed
  + Trap is a software generated interrupt caused by error or user request
  + OS determines which type of interrupt has occurred by polling or the vectored interrupt system
* Device-status table:contains entry for each I/O device indicating its type, address, and state
  + OS indexes into the I/O device table to determine device status and to modify the table entry to include interrupt
* A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage. Typically, operating systems have a device driver for each device controller. This device driver understands the device controller and presents a uniform interface for the device to the rest of the operating system.
* Direct memory access (DMA) instructs the device controller to move data between the devices and main memory. Efficient because it removes the CPU from being responsible for transferring data.
* Storage structure:
  + Main memory – random access, volatile. Any power loss to system will result in erasure of the data stored within that memory, thus not suitable for permanent storage or backup
  + Secondary storage – extension of main memory That provides large non-volatile storage
  + Disk – divided into tracks which are subdivided into sectors. Disk controller determines logical interaction between the device and the computer.
* Physical memory is the memory available for machines to execute operations (i.e., cache, random access memory, etc.). Virtual memory is a method through which programs can be executed that requires space larger than that available in physical memory by using disk memory as a backing store for main memory. Logical memory is an abstraction of the computer’s different types of memory that allows programmers and applications a simplified view of memory and frees them from concern over memory-storage limitations.
* Caching – copying information into faster storage system
* Multiprocessor Systems: Increased throughput, economy of

scale, increased reliability

* + Can be asymmetric or symmetric.
  + Clustered systems – Linked multiprocessor systems.

High-availability due to redundancies in the case of a failure.

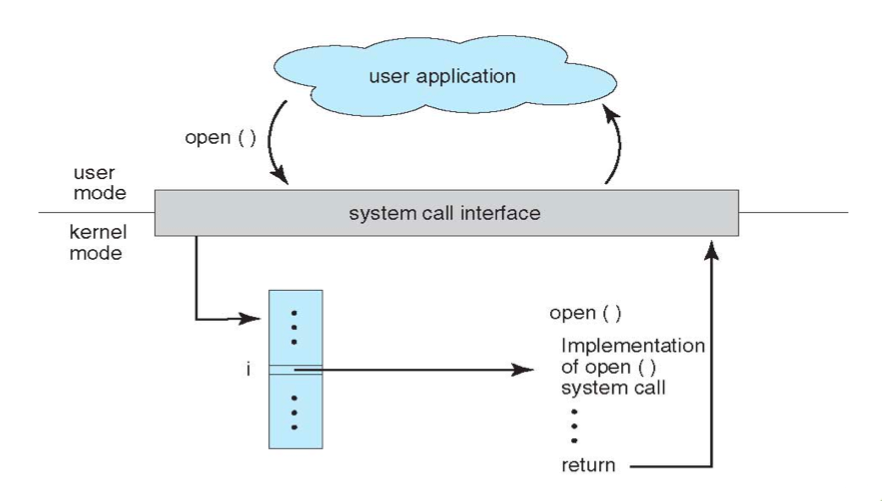
* + multi-core processing
* Multiprogramming – Provides efficiency via job scheduling
  + When OS has to wait (ex: for I/O), switches to another job
* Timesharing – CPU switches jobs so frequently that each user

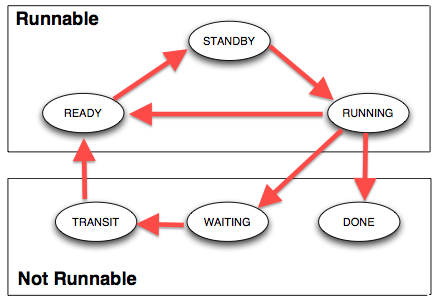
can interact with each job while it is running (interactive computing)

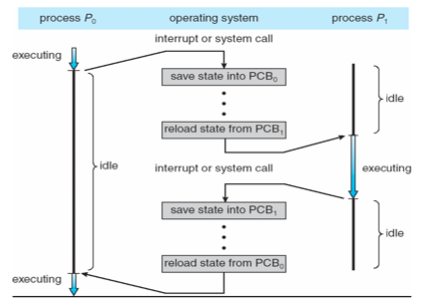
* Single-threaded processes have one program counter, multi-threaded processes have one PC per thread
* Protection – mechanism for controlling access of processes or users to resources defined by the OS
* Security – defense of a system against attacks
* User IDs (UID), one per user, and Group IDs, determine which users and groups of users have which privileges
* Cloud computing is a type of computing that delivers computing, storage, and application services across a network.

Potentially multiple choice question about system calls

**Ch.2 – OS Structures**

* User Interface (UI) – Can be Command-Line (CLI) or Graphics User Interface (GUI) or Batch
  + These allow for the user to interact with the system services via system calls (typically written in C/C++)
  + Command interpreter:
    - Upon the user issuing a command, the interpreter jumps to the appropriate section of code, executes the command, and returns control back to the user;
    - The interpreter loads the appropriate system program into memory along with the appropriate arguments
* Other system services that are helpful to the user include: program execution, I/O operations, file-system manipulation, communications, and error detection
* Services that exist to ensure efficient OS operation are: resource allocation, accounting, protection and security
* Most system calls are accessed by Application Program Interface (API) such as Win32, POSIX, Java
* Usually there is a number associated with each system call
  + System call interface maintains a table indexed according to these numbers
* Parameters may need to be passed to the OS during a system call, may be done by:
  + Passing in registers;
  + Parameters stored in a block; the address of the block passed as parameter in a register
  + Pushed onto the stack by the program and popped off by the OS
  + Block and stack methods do not limit the number or length of parameters being passed
* Process control system calls include: end, abort, load, execute, create/terminate process, wait, allocate/free memory
* File management system calls include: create/delete file, open/close file, read, write, get/set attributes
* Device management system calls: request/release device, read, write, logically attach/detach devices
* Information maintenance system calls: get/set time, get/set system data, get/set process/file/device attributes
* Communications system calls: create/delete communication connection, send/receive, transfer status information
* Goals in OS design:
  + User goals: easy to learn and use, reliable, safe, and fast;
  + System goals: easy to design, implement, and maintain; flexible, reliable, error-free, and efficient.
* OS microkernel approach:
  + Easy to extend. New services added to user space and thus no need to modify the kernel.
  + More security and reliability since most services are running as user processes.
  + Increased system function overhead.
* OS 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
* OS modular approach:
  + The modular approach combines the benefits of both the layered and microkernel design techniques. In a modular design, the kernel needs only to have the capability to perform the required functions and know how to communicate between modules. However, if more functionality is required in the kernel, then the user can dynamically load modules into the kernel. The kernel can have sections with well-defined, protected interfaces, a desirable property found in layered systems. More flexibility can be achieved by allowing the modules to communicate with one another.
* Virtual machine: uses layered approach, treats hardware and the OS kernel as though they were all hardware.
  + Host creates the illusion that a process has its own processor and own virtual memory
  + Each guest provided with a 'virtual' copy of the underlying computer
* Application failures can generate core dump file capturing memory of the process
* Operating system failure can generate crash dump file containing kernel memory

**Ch.3 – Processes**

* Process contains a program counter, stack, and data section.
  + Text section: program code itself
  + Stack: temporary data (function parameters, return addresses, local variables)
  + Data section: global variables
  + Heap: contains memory dynamically allocated during run-time
* Process Control Block (PCB): contains information associated with each process: process state, PC, CPU registers, scheduling information, accounting information, I/O status information
* Types of processes:
  + I/O Bound: spends more time doing I/O than computations, many short CPU bursts
  + CPU Bound: spends more time doing computations, few very long CPU bursts
* The possible states of a process are: new, running, waiting, ready, and terminated. The process is created while in the new state. In the running or waiting state, the process is executing or waiting for an event to occur, respectively. The ready state occurs when the process is ready and waiting to be assigned to a processor and should not be confused with the waiting state mentioned earlier. After the process is finished executing its code, it enters the termination state
* When CPU switches to another process, the system must save the state of the old process (to PCB) and load the saved state (from PCB) for the new process via a context switch
  + Time of a context switch is dependent on hardware
* Parent processes create children processes (form a tree)
  + PID allows for process management
  + Parents and children can share all/some/none resources
  + Parents can execute concurrently with children or wait until children terminate
  + fork() system call creates new process
    - exec() system call used after a fork to replace the processes' memory space with a new program
* The short-term scheduler is designed to frequently select a new process for the CPU, at least once every 100 milliseconds.
* The long-term scheduler executes much less frequently; minutes may separate the creation of one new process and the next. The long-term scheduler controls the degree of multiprogramming.
* Cooperating processes need interprocess communication (IPC): shared memory or message passing
* Message passing may be blocking or non-blocking
  + Blocking is considered synchronous
    - Blocking send has the sender block until the message is received
    - Blocking receive has the receiver block until a message is available
  + Non-blocking is considered asynchronous
    - Non-blocking send has the sender send the message and continue
    - Non-blocking receive has the receiver receive a valid message or null
* A loopback is a special IP address: 127.0.0.1. When a computer refers to IP address 127.0.0.1, it is referring to itself. When using sockets for client/server communication, this mechanism allows a client and server on the same host to communicate using the TCP/IP
* Data can be represented differently on different machine architectures (e.g., little-endian vs. big-endian). XDR represents data independently of machine architecture. XDR is used when transmitting data between different machines using an RPC (remote procedure call).
* Marshalling involves the packaging of parameters into a form that can be transmitted over the network. When the client invokes a remote procedure, the RPC system calls the appropriate stub, passing it the parameters provided to the remote procedure. This stub locates the port on the server and marshals the parameters. If necessary, return values are passed back to the client using the same technique.
* Binding of client and server ports for RPC can be either at compile time using predetermined port address, or dynamically through rendezvous mechanism.

Went over fork example again

The progran itself is a fork, including what is in it

PROGRAM{

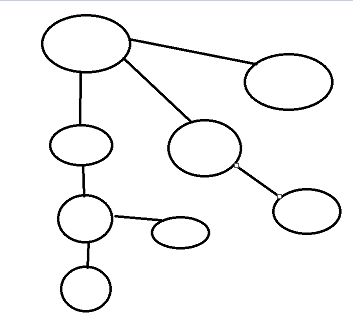
Fork

Fork

fork

}

8 processes



Pstree -Apn | grep PROGRAMNAME to see process tree

If in the child process, returns 0

If in parent process returns child pid?