Comp 362 Midterm 1 Study Guide

Three main goals of an operating system	- Make user problems easier to solve - Have the computer system be more convenient to use - Efficiently use hardware
Role of OS as resource allocator	Manage resources and conflicts to ensure fair and efficient use
Role of OS as control program	Controls how programs are executed to avoid errors
What is an OS kernel?	Communicates between the computer and the hardware. It runs at all times on the computer
Purpose of interrupts	To allow hardware to tell the computer that an event that needs attention has occurred (**attention w***)
Differences between trap and interrupt	**Interrupt** - hardware says to the computer, "I need attention and need it now!" **Trap** - is more chill. Only occurs **(while strolling through the park)** when the computer encounters an error or user request
Can traps be generated intentionally by a user program? If so, for what purpose?	Can be generated intentionally when calling OS routines and fixing arithmetic errors
Describe synchronous and asynchronous communication (I/O) between OS and device	**Synchronous** - is like a stop sign; control is returned to the user program ONLY when I/O **(the old lady crossing the street)** has finished **Asynchronous** - doesn't wait **(like that a****** that weaves in and out of traffic);** but performs a system call which requires OS to perform the restricted operation and check device-status table **(he's like checking his blinds now)** which has entries for type, address, and state for each I/O device
Two reasons why **caches** are useful	1. Useful when two components need to exchange data **(saliva)** but have different transfer speeds 2. When a fast device wants to find data, it doesn't need to wait for the slower device **(like how big dogs run way faster than**

	their owners)
What problems do caches solve?	Solves transfer problems by creating an immediate buffer between the two components to allow for faster-transferring
Problems that caches cause	Must always be updated with the most recent version of the data when multitasking
Why can't you make the cache as big as the device it's caching?	It's pricey **(like spoiled brats)** and has to still be able to retain data even when electricity is lost
Challenges in using cached data	- Cache is smaller **(like Annah ;)** than the storage being cached, so only a limited amount of data can be stored - Must always be updated with the most recent version of the data when multitasking
What is **multi-programming**?	A form of parallel processing in which several programs can run on the processor at the same time; Programs are loaded in main memory and ready to execute
Why is multi-programming used in modern computers?	Maximizes use of CPU because another job can execute whenever an interrupt occurs
Explain what is meant by stating that modern OS is **interrupt-driven**	Means that software will react appropriately whenever hardware encounters an interrupt **(like GOOD parents when kids have temper tantrums)**
How does the distinction between kernel mode and user mode function act as a **rudimentary form of protection (security) system**?	- **Kernel-mode** allows for control determining if interrupts should be enabled/disabled -Since the CPU has limited capability in user mode, it can better enforce the protection of critical resources
What is a **system call**?	- Used for under-the-hood operations **(mafia deals)** - Can authenticate request source and check it has authority to perform requested operation - Can change to kernel mode to perform the privileged operation - Executes return to reset back to user mode
Different between **process** and **program**	**Process** - program in execution; passive entities **Program** - has instructions that perform a task; active entities

	Processes depend on programs to tell them what to do **(their b****)**
What is **system concurrency**?	Gives the appearance of simultaneous execution by interleaving processes; Does this by multiplexing one or more CPUs

Process management activities of OS	creating and deleting user and system processes; stopping and resuming processes, providing mechanisms for process synchronization, process communication, deadlock handling
What is **memory management**?	States what is in memory and when
Main memory management activities	Keeps track of parts of memory that are currently being used; Decide which processes and data should move in and out of memory; Properly allocates and deallocates memory
What is a **VM**?	Allows guest OS to run on host OS by treating hardware and OS kernel as if they were hardware **(mi casa es su casa)**; Physical computer's resources are shared when one is created
2. What is a **bootstrap program**?	A program that initializes the OS during startup from a fixed memory location in NVRAM that holds code provided by the computer vendor **(firmware)**; Knows how to load OS and how to start executing system by locating OS loader and loading it into memory
What is an **OS loade**r?	An **OS loader** is responsible for loading programs and libraries and is one of the essential stages in the process of starting a program.
What are the differences between a **bootstrap program** and an **OS loader**?	The **bootstrap** tests initialize the hardware and then start the **OS loader** so that the user can choose from different OSes. So bootstrapping works independently and the OS loader is only an intermediary loader.
What are the **services provided by an OS** to users?	**User Interface, Program Execution, I/O operations, File-system manipulation, Communications, and error detection**.

How does **OS program execution provide convenience** to the user?	The OS provides an environment where users can conveniently execute programs in an efficient manner by not having the user worry about memory allocation, or loading the program into ram since it's all taken care of by the OS.
How do an OS's I/O operations provide convenience to the user?	each program requires input and after processing the input, it produces an output which is the job of I/O devices that run hidden from the user.
How does the **OS file-system manipulation** provide convenience to the user?	usually, a user is required to manipulate various types of files **(opening, saving, deleting files)** and this can be performed by the OS.
How does the **OS's communication provide convenience** to the user?	The OS performs communication between different processes in shared memory. In a multi-tasking environment, it's important to have the processes communicate and exchange information that the OS takes care of.
How does the OS's error detection provide convenience to the user?	The OS constantly monitors the system for detecting the error and fixing them which would be too critical to be handed over to the user programs.
How does an OS user interface provide convenience to the user?	It's used in almost all operating systems and is important in communicating between the user and the hardware.
Why shouldn't user-level programs should provide OS services?	because the services deal with what the OS system should do when facing a situation where the system is at risk. this needs to be handled in a specific way that a user program should not interfere with. **(Bottom line is that computers can't trust humans to allocate resources, for example, they may allocate too much)**
Purpose of **command interpreter (shell)**	used to fetch command from a user and execute it
Why command interpreter (shell) is usually separate from the kernel? **(Hint: it's shy... not really)**	Command interpreter allows users to directly enter commands they want the OS to perform. Because of this, it's subject to change so it has to be separate from the kernel.

OS services for resource sharing	1. **Resource Allocation** is the service for when multiple users/jobs run concurrently and resources must be allocated to each of them. 2. **Accounting**, service to keep track of which users use how much and what kinds of computer resources. 3. **Protection & Security**, a service where the owners of information stored in a multi-user computer system could want to control the use of that information, and concurrent processes shouldn't interfere with each other. Protection is a mechanism and security is a policy **(both control access but security requires user authentication i.e. security needs to have policies to ensure the right ppl have access)**
What are **system calls** used for?	a pragmatic way in which a computer program requests a service from the kernel of the OS.
How can **system calls** be implemented?	There is a number associated with each system call, and the system call interface maintains reference table to system call handlers that are indexed according to the associated numbers. Then the system call interface invokes the intended system call in the OS kernel and returns the status of the call and any return values.

Do application programs usually use system calls directly?	no, because they usually don't have direct access to system calls, but they can access them through an API (Application Program Interface)
How can **parameters** be passed to **system calls**?	**THREE GENERAL METHODS:** 1. simplest way; pass parameters in registers, though there sometimes might be more parameters than registers. 2. parameters are placed, or pushed, onto the stack by the program and popped off the stack by the OS. 3. parameters are stored in a block, or table,

	in memory, and the address of the block is passed as a parameter in a register.
Common types of system calls and what they're used for	- **Process Control:** running program needs to be able to stop execution (normally/abnormally) -**File Management:** some common system calls are to create, delete, read... and there is a need to determine the file attributes; get and set file attributes -**Device Management:** user programs request the device and when finished they release the device; similar to files, users can read, write, and reposition the device -**Information Maintenance**: for system calls purely for transferring info between the user program and OS (like time or date); the OS keeps info about all process and uses system calls to report this info. -**Communications:** there are 2 models of interprocess communication: message-passing and shared memory; message uses a common mailbox and memory uses system calls to get access to memory
Purpose of system programs **(utilities)**	provide a convenient environment for program development and execution is and provide basic functionalities to the user
Why are **system programs a better solution** than providing system/shell built-in functions?	it can provide a convenient environment for program development and execution. **Yes**, and they are used so that users don't have to write new programs to perform basic functions
List the **categories of system programs** that you know and describe what they are used for	-**File management**: manipulate files and directories - **Status Information:** multiple things that it can do (like a lot), such as asking the system for info, provide detailed performance, logging, and debugging info, format and print output to terminal or output devices, implement a system database to sore and retrieve configuration information, or provide

	utilities to manipulate per-program files -**File Modification:** text edit to create and edit files, commands to search contents of files or transform the text -**Programming** - language support: compilers, assembles, debuggers, and interpreters -**Program loading and execution:** absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level & machine language -**Communications:** provide a way to create virtual connections between processors & computer systems
The main advantage of the **microkernel approach** to system design	it provides minimal process and memory management with communication facility; more reliable **(because less code running in kernel)** and secure
Disadvantages of the **microkernel** approach to system design	suffers from performance decrease due to increased system function overhead
Pragmatic approaches that employ the microkernel ideas but limit the deficiencies of pure microkernel-based architectures?	Implement kernel modules so that each component is loaded into the kernel when it is needed
3. What is a **PCB (Process Control Block)**?	it's a block that contains info about each process **(process state, program counter, CPU registers, etc)**
How **PCB (Process Control Block) is used by OS**	The OS maintains a list of PCBs for all processes like arrays, linked lists, and hybrids.
What information does PCB (Process Control Block) contain?	Process state, program counter, CPU registers, CPU scheduling info, memory-management info, accounting info, and I/O status info.
How do processes migrate between a variety of OS queues?	pointers to PCBs are used to migrate between queues **Job queue -** all processes are already in the system **Ready queue -** processes are finishing for CPU usage **Device queue -** OS determines which

	process belongs to the device and waits for that I/O device
How does the **state of process** change and under which circumstance?	-When the process is "new" it is being created -When the process is "ready" it is waiting to be assigned to a processor -When the process is "running" the instructions are being executed -When it's "waiting", it's waiting for some event to occur. -when it's "terminated", the process is finished executing. **(If interrupts occur, it goes back to "ready", then "waiting" until the operation is finished and back to "ready")**
In what state are process instructions being actively being executed by the CPU?	"running" state
What is **blocking communication**?	a type of **synchronous** message passing where blocked processes wait in the waiting queue **(processes being blocking send and blocking receive: AKA you can't get the data and you'll need and have to wait for it Basically when a process waits for an event)**
What is **non-blocking communication**?	a type of **asynchronous** message passing where the send and receive don't go in the waiting queue; non-blocking send sends a message and continues; non-blocking receive receives a message or null **(A process doesn't need to wait for an event)**
What is a **rendezvous**?	communications between blocking sender and receiver
What is **direct interprocess communication**?	each process that wants to communicate has to explicitly name the sender of the communication
What is **indirect interprocess communication**?	messages are sent to and received from mailboxes

How are process creation and termination	**Creation**: parent process creates children

handled by OS?	processes, and so on, making a hierarchy of processes **Termination**: process executes the last statement and asks the OS to delete it
What is the **producer-consumer paradigm**?	it is a common paradigm for cooperating processes where the producer process makes information that is consumed by the consumer process
Differences in using **bounded** and **unbounded buffers** in the communication link	An **unbounded** buffer has no practical limit on the size of the buffer, while a **bounded** buffer does have a fixed size
Short-term scheduling	selects which processes should be brought into the ready queue
Medium-term scheduling	used for swapping processes out and in
Long-term scheduling	aka CPU scheduler; selects which process should be executed next and allocates CPU
Actions the kernel takes to context-switch between processes	when a clock interrupt happens, the OS saves the PC and user stack pointer of the currently executing process and transfers control to the kernel clock interrupt handler. The handler saves the rest of the registers to the process PCB. The OS invoked the scheduler to determine the next process to execute. Finally, the OS retrieves the state of the next process from its PCB and restores the registers.
Four common uses of **IPC (Inter-Process Communication)**	Information sharing (for files and shared memory); Computation speedup (for multi-core processors); Modularity (for software engineering); and Convenience (for signaling events in parallel activities)
Main mechanisms for IPC (Inter-Process Communication)	message passing (through signals and pipes), where the kernel actively facilitates the communication, and shared memory, where there is no kernel involvement after the shared space is allocated
Differences between **process** and **thread**	**Process** - an executable entity with process attributes (process ID, process group ID, user ID, group ID) **Thread** - scheduling entity that needs stack, PC, and different registers

Does every process have a thread?	Yes
Minimum and maximum threads per process	Trick question. There is none. :P
Can a thread span multiple processes?	Nope
Why use threads?	- Allow continuous execution if part of the process is blocked - Resource sharing is easier because threads have access to global processes - Faster to work with threads than processes (similar to above) - Thread switching requires less overhead than process switching - Threads can run in parallel if there are multiple cores
Symmetric Multiprocessing (SMP)	- One kernel runs on all cores - Cores used uniformly
Asymmetric Multiprocessing (ASMP)	- One kernel per core - Kernels can be on different OS - Use of each core is specialized
Which of the following components of program state are shared across threads in a multithreaded process, and which ones are not? - Register values - Heap memory (dynamic allocation) - Global variables - Stack memory(call stack)	- Global variables are shared because they're stored in heaps and heaps are shared with threads - The rest aren't shared because each thread has its own stack, PC, and registers
User-level threads	- Done by user-level thread libraries; - Calls user-space code but can also call the kernel
Kernel-level threads	Only runs on kernel and never the user code
Ways in which a user thread library can be implemented using kernel threads	**Many-To-Many** - many user threads mapped to many kernels **One-To-One** - each thread mapped to each kernel **Many-To-One** - user threads are all mapped to a single kernel

What is a **signal**?	A way of communicating issues **(Hello, I'm signal 1)**

Issues with signals targeted at **multi-threaded processes**	Can be mixed amongst one another in multi-threaded system
Options OS designers have regarding the problem with signals	- Deliver signals to the thread where the signal occurred - Deliver signals to every thread in the process - Only deliver signals to certain threads - Have designated thread receive all signals
How does **POSIX** solve signals in multi-threaded processes?	Signals go to the process, not the thread. So threads may share signal handlers **(kinda like how they share globals)**
What is a **thread pool**?	Where threads go to wait for work **(their break room)**
Advantages of using a thread pool	Faster than having to create new threads to get the job done
Three types of thread pools	**Single thread executors** - pool size of one **Fixed thread executors** - fixed pool size **Cached thread executors** - no predetermined size limit
You are asked to develop a **Web server that responds** to many users accessing it at the same time. Obviously, you choose to design it with threads, as each thread can be committed to serving one client, so the others do not have to wait. Do you think that using a pool of threads would be proper in this context? Which type would be best?	Cached pool, to account for the fluctuations of requests throughout the day (less at night and a little more during the day)
Why threads may need private data	To protect information **(Thread police! Assuming they actually do their job...)**
Name an application that may have threads containing private data	Credit card transaction
Consider developing two applications: 1. A program that takes several input parameters, computes a complicated formula that requires a lot of CPU time, and then outputs the results to the screen. 2. A collaborative graphical design tool that takes input from several graphical pads,	1. this is a **no-no**. Would have to take constantly update info to ensure consistency 2. **Yes**, so that tools could work with one another

mice, and keyboards, from local users as well as other people on the network, displays their ideas on the screen constantly rendering the view to reflect the new input. In which program would you use multiple threads knowing that the target architecture has one (core) processor?	