MBR(Master boot record)-Sector 0 of the disk, used to boot computer, end of MBR contains partition table. Boot block-The first block on the active partition, contains program that loads the operating system. Superblock-Conains all key parameters about the file system and is read into memory when the computer is booted or the file system is first touched. Contiguous-file is consecutive blocks, good because files can be read in a single operation, leads to fragmentation though. Linked list allocation-each block is a node with a pointer to the next, makes random access very slow. I-node - Array stores the addresses of a files attributes and a list of all its blocks. Hard Link-A link is created from the name specified to the existing file that you want the link to go to. Soft link-Points to a tiny file naming another file. Less efficient but has the advantage of being able to cross disk boundaries and even name files on remote computers. Journaling file system-Keeps log of what the file system is going to do before it does it, if there is a crash it can look at this journal to see what the planned work was and finish the job. Logged operations need to be things that can be repeated as often as necessary without causing harm. Virtual file system-Abstract out the part of the file system that is common to all file system and put that code in a separate layer that calls the underlying concrete file systems to actually manage the data. System calls related to files go to the virtual file system which then makes the appropriate file system call. Block size-If blocks are to big small files will take up all the space, if blocks are to small it takes longer to retrieve files due to multiple seeks and rotational delays. Free blocks-One method is linked list of disk blocks, each block holding as many free disk block numbers as it can fit. Other method is bitmap, which uses 1 bit per block, uses 1 for free, 0 if used. Backup procedures- Physical dump: dumps entire disk onto output tape in order, can be made almost 100% bug free. Problem is if bad blocks are visible to the operating system then it could result in endless disk read errors while trying to backup the bad-block file. Logical dump: starts at one ore more specified directores and recursively dumps all files and directories found there that have changed since some given base date. Makes it easier to restore specified files or directories on request. Dumps all directories and files modified since the specified date, also dumps unmodified directories on the path to the modified file or directory in order to make it possible to restore the dumped files and directories to a fresh file system on a different computer. Incremental dump-Make complete dump periodically and make a daily dump of files modified since full dump. File system consistency-If a set of blocks are modified and the system crashes before all have been written out this will cause inconsistency especially bad if those are i-node, directory, or free list blocks. To check for block consistency the program builds two tables, one keeps track of how often a block is present in a file, another keeps track of the block in the free list. If blocks are consistent then there will be a 1 one table, and a 0 in the other. If block has 0 for both it is added to the free list. If a block appears multiple times in the free list then the free list is rebuilt. If the same block is present in 2 or more files, this is handled by copying the contents of block 5 into a new block, and inserting the copy into one of the files, then report this to the user. Caching-Pure LRU is not used because blocks like i-node blocks, indirect blocks, directory blocks, and full data blocks will probably not be needed soon and are sent to the front so they can be rewritten, this also helps consistency because the longer those modified blocks are in the cache the more likely they are to be lost to a crash. Write through cache-Modified blocks are written back to the disk immediately. Write back cache-Blocks are written back to disk periodically, such as maybe every 30 seconds. Reducing disk arm motion - move blocks that are likely to be accessed in sequence close to each other, put i-nodes in the middle of the disk so that they are more likely to be closer to the first block of the file. Unix V7-For very large files uses i-node with single indirect block, or double indirect block ,or triple indirect block. To look up file the file system first locates the root directory, then looks up the i-node number of the first component of the path, locates the directory of the first component in the path, then looks up the next component. Block device-Stores information in fixed size blocks, hard disks, cd-roms, and usb sticks are examples. Character device-delivers or accepts streams of characters, printers, network interfaces, mice, and most other devices that are not disk like can be seen as character devices. Device controller-The electronic component of an I/O device. Memory mapped I/O-Map all control registers into memory space, each control register is assigned au nique memory address to which no memory is assigned Memory mapped I/O is good because device control registers are just variables in memory, without memory mapped I/O you need to use assembly code. Also memory mapped I/O does not require any special protection mechanism to keep users from performing I/O. Also every instruction that can reference memory can reference control registers. The downside is you have to avoid caching control registers because if they are in the cache then the software would just keep reading them from the cache and not have any idea if the device was actually ready. port mapped I/O-Each control register is assigned an 8 or 16 bit I/O port number that is protected so user programs cannot access it. DMA(Direct memory access)-CPU programs the DMA controller by settings its registers so it knows what to transfer where, DMA controller initiates the transfer by issuing a read request, then the write occurs, and the disk controller sends an acknowledgement signal to the DMA controller. Steps 2 through 4 are repeated until the byte count reaches 0. Programmed I/O-I/O done by having the CPU do all the work. While the printer is printing the CPU polls the device over and over to determine when the device is ready to accept another character. Interrupt driven I/O-The printer generates an interrupt when it has printed a character and is ready to accept another, this allows the CPU to do work in between. I/O using DMA-The DMA feeds the printer characters so that CPU time is not wasted on I/O, this is basically programmed I/O with the DMA doing the work instead of the CPU. Cycle stealing-Device controller sneaks in and steals an occasional bus cycle from the CPU. Burst mode-DMA controller tells the device to acquire the bus, issue a series of transfers, then release it. fly-by-mode-The DMA controller tells the device controller to transfer the data directly to main memory. Word-at-a-time-After each word is transferred the DMA controller decides which device to service next. Precise interrupt-An interrupt that leaves the machine in a well-defined state, pc counter is saved to a known place, all instructions before the one pointed to by the PC have been fully executed, No instructions beyond the one pointed to by the PC have been executed, The execution state of the instruction pointed to by the PC is known. Imprecise interrupt-the operating system write has to figure out what has happened and what still needs to happen. Goals of I/O software-Write programs that can access any I/O device without having to specify the device in advance, name of a file should not depend on the device in anyway, errors should be handled as close to the hardware as possible. Uniform interfacing for device drivers-If device drivers look the same to the operating system then the O/S doesn't need to be modified when a new device comes along. Buffering-Unbuffered requires the process to run alot which is inefficient, buffering in user space only wakes the process when the buffer is full but the buffer could be paged out when a character arrives, this can only be prevented by locking the buffer in memory, buffer inside the kernel - When the buffer is full page from user buffer is brought in, problem is characters that arrive while the page with the user buffer is being brought in there is no place to put them, this requires a 2nd kernel buffer. Low level format-A series of concentric tracks each containing some number of sectors, with short gaps between the sectors. Has preamble to show start of sector, data, and then ECC with redundant information to recover from errors. Partition formatting- after low level formatting the disk can be divided into partitions which logically are treated as seperate drives. High level format-lays down boot block, free storage administration, root directory, and an empty file system. Cylinder skew-The sector 0 position on each track is offset so that the disk can read multiple tracks in one operation. Interleaving-Gives controller breathing space between consecutive sectors in order to copy the buffer to main memory. Stable storage-Uses a pair of identical disks with corresponding blocks working together to form one error-free block. Stable write - Write, then read to verify, then write again if wrong, after n failures remapped onto a spare and repeats till the write suceeds. Then the same is repeated on drive 2. Stable reads read, then check ECC, repeat if wrong, try n times, if failure still read from drive 2. Crash recovery-Scans drives, if one block has an ECC error then the program replaces it with the matching block from the other drive. raw(non-canonical) input-Drivers job is to pass input forward unmodified, program gets a series of ASCII codes from keyboard. cooked(canonical) input-Drivers handles intraline editing and delivers corrected lines to the user programs. Escape sequence-Commands in an output driver to move the cursor, insert and delete characters or lines at the cursor and so on. termcap-terminal database that defined a number of basic actions such as moving the cursor to(row, column). This allowed software to be more easily written to work on different terminals. X-window system-portable and runs entirely in user space, split into client software and host software which can run on different computers, the server actually runs on the users computer and the client can be run on a remote computer server because servers job is to display bits on the users screen. Power management-Software can be programmed to only access the hard drive during time frames specified by the OS so that the disk will only run during those time frames. When using a multimedia viewer that displays a fixed number of frames per second the program can use the CPU for a period, then shut down, then start up again when its time to process the next frame. Event driven-Flow of program determined by user. procedural driven-flow of program determined by programmer.

**The 60 was supposed to be 60% I/O wait (actually, whether sequential or parallel): 1 minute \* 0.6 is 36 seconds. That means it’s using the CPU for (60 seconds - 36 seconds) or 24 seconds out of every minute. Divide 10 min by 24 seconds and you get the sequential execution time of 1 job. Then multiply by 2 for two jobs running in sequence. The parallel time is based on the utilization formula 1 – percent wait^number of processes. Since they're running in parallel, you don't need to multiply by 2 for that one.**

Extended machine-OS abstracts interfaces to coder

Resource Manager-OS controls everything itself

**There are four principal events that cause a process to be created: System initialization. Execution of process creation system call by running a process. A user request to create a new process. Initiation of a batch job.**

The value of a semaphore is the number of units of the resource which are free. (If there is only one resource, a "binary semaphore" with values 0 or 1 is used.) The P operation busy-waits (uses its turn to do nothing) or maybe sleeps (tells the system not to give it a turn) until a resource is available, whereupon it immediately claims one. The V operation is the inverse; it simply makes a resource available again after the process has finished using it. The P and V operations must be atomic, which means that no process may ever be preempted in the middle of one of those operations to run another operation on the same semaphore.

**Microkernel-smaller/lightweight/portable/trusted codebase**

**Monolithic-more abstract/highly controllable**

Page replacement:

NRU: when a page is refer/mod the r/m bit is set-4classes

2nd chance: FIFO + r-bit check if r=1 page put to bottom

Clock: FIFO with no reordering by linked list

LRU: linked list has of all pages. Back=lru, front=mru

NFU: pages have incremented counters tracking their use

**CPU time to load page table: page size => bits (2^x),2^(address space – page size) = number of pages. Number of pages \* time to load = time to load pages**

Pthread\_join(x) makes calling process wait for x thread

**Page table size: Single level: 232 virtual address space mapped to 4KB (212 bytes) pages: the number of virtual pages is 220 (= 232 / 212). However, if the page size is increased to 32KB (215 bytes), only 217 pages are required.**

**Multi-level: first table: 2^PT1 + 2^PT2**

Real-time scheduling: while sum(actual/period) <= 1

pre-emptible resource-one that can be taken away from the process owning it with no ill effects. non-premptible-something such as a cd-reocrder being taken away from a CD-rom burning a cd. Deadlock conditions-each resource is currently assigned to exactly one process or is available, processes currently holding resources that were granted earlier can request new resources, resources previously granted cannot be forcibly taken away from a process, they must be explicitely released by the process holding them, there must be a circular chain of two or more processes each of which is waiting for a resource held by the next member in the chain. Deadlock-A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Vector graphics-Use of geometrical primitives such as points, lines, curves, and shapes which are all based on mathematical equations to represent images

bitmap-a data structure representing a generally rectangular grid of pixels, or points of color. TrueType-An outline font standard originally developed by apple computer in the late 1980's.

(E1,E2...Em) (A1,A2..A3)

[C11 C12...C1m] [R11,R12...R1m]

[Cn1 Cn2...Cnm] [Rn1,Rn2...Rnm]

Look for unmarked process Pi for which the i-th row of R is less than or equal to A

If such a process is found add the i-th row of C to A, mark the process, and go back to step 1

when done unmarked processes are deadlocked

The memory manager allocates memory to processes, keeps track of memory in use, and deallocates memory when processes finish. Without abstraction:u(user program at top, operating system at bottom of ram),(operating system in rom, user program in ram), (devices drivers in rom, user program, operating system in ram). IBM 360 had memory divided into blocks with a 4 bit protection key. The OS trapped any attempt by a running process with access memory with a protection code different from the PSW key. Drawback was that programs referencing absolute physical memory could still interfere with each other. IBM fixed it by relocating other programs when a new program loaded by adding the programs address to the addresses of other programs in memory.Purpose of address space gives each process its own address space in which to work, preventing programs from interfering with each other. Relocation problem is how to give a process its space so location 28 in one process is different from location 28 in another. The solution was to add base registers which hold base address and limit register which holds the processes limit. When a process tries to access a location in memory the base address is added to the address generated by the process and makes sure the address offered is not greater than the limit. Memory overload strategies: swapping-bring each process in its entierety, run for awhile, put back on disc. virtual memory-allows programs to run even when they are only partially in main memory. Processes are periodically swapped out to disk to allow other processes to run. In its most simple form swapping creates holes in memory. Memory compaction moves all processes down as far as possible to eliminate holes. Linked list memory management: First fit:Find first hole big enough for process, next fit: like first fit but starts where the last check left off. Best fit:search hole list, find smallest hole that can be used. Worst fit-Takes largest available hole. Quick fit-maintains seperate lists for commonly requested sizes. Bit map memory management-memory divided into allocation units, bitmap represents each with a bit, 0 means free, 1 means occupied. Virtual memory-Program broken into pages, when the program references address space not in memory, the OS gets the missing page and re-executes the instruction that failed. Virtual addresses go to the MMU which maps them onto physical addresses Page Frame:The unit in physical memory corresponding to pages. Page fault-Occurs when the program tries to use a page thats not in memory, causing the OS to go get the page from disk. Page replacement algorithm: optimal-Each page labeled with the number of instructions until it is used, one that won't be used for the longest time is removed. Not Recently Used-R set when referenced, m when written. Second-Chance Replacement-Looks for R bit that is 0 and replaces page. Any R bits that are 1 and encountered along the way are set to 0. Clock replacement-like second chance but starts at the oldest page, which the "hand" points to. Not frequently used-each page has a counter, R is added to the counter at each clock interrupt. The page with the lowest counter value is replaced. Aging algorithm-At each clock tick, the counter is shifted right and the new MSB is the value of R. Demand paging-Pages are loaded only when the process tries to use them. Working set-The set of pages that a process is currently using. Thrashing-A program causing page faults every few instructions. Locality of reference-During a phase of execution a process references only a relatively small fraction of its pages. Prepaging-Loading pages before letting processes run. Internal fragmentation-When part of a page is left empty which wastes space. Ispace and Dspace doubles the available address space. Shared pages-When two processes share some code if the scheduler removes a process from memory, the other process will generate a large number of page faults to bring those pages back. When data is shared and a program modifies data the operating system makes a copy of the page so that each process has its own copy of the page. Paging daemon-Awakens periodically to inspect memory and evict pages if there are not enough free pages. Instruction backup-A hidden register that stores the page counter so that an instruction can easily be restarted after a page fault. Segments-Many completely independent address spaces. different segments can have different length and length can be increased when something is pushed on the stack and decreased when something is popped off the stack. Segments are usually large enough that they are not filled up. External fragmentation-When a segment is evicted and a smaller segment is put in its place a hole develops, this is fixed by compacting the segments so that the holes are combined into one large free memory space. File-Logical unit of information created by processes. Peristance-information in files not effected by process creation and termination. File system-the part of the operating system that deals with files, this includes how they are structured, named, accessed, used, implimented and managed. File structures-byte sequences, record sequences, record tree

operating systems provide programmers with a clean abstract set of resources instead of messy hardware ones and manage hardware resources. Extended machine-Provides clean interface for programmer or resource manager-manages system resources. Virtual address-Virtual addresses point to a processes address space, physical addresses point to real memory. MMU maps it to physical address. When virtual memory is not used it will be the same as the physical address. Physical address always goes directly to the memory bus. Process-A program in execution, has address space which are memory locations the process can read and write. Contains program, data, and its stack and a list of resources. Deadlock-Two programs try to access the same resources, each program ends up with permission to use a portion of the resources, programs are locked since the other resources they need to finish are being used by the other program. System calls-used to obtain services from the operating system. fork-create a child process identical to the parent. waitpid-Wait for child process to terminate. execve-Replace a processess core image. Windows decouples the name of the system call and the API function so that system calls can be changed without invalidating programs using the old system calls. Process creation- System initialization, processes are created when a running process executes a process creation system call, user requests to create a new process, a batch job is initiated. process termination-a normal exit, an exit caused by an error, a fatal error, another process kills it. Process states-running, meaning it is using the CPU, Ready-Waiting for another process to run, Block-Waiting for some external event before it can run. Thread- 2 more more concurrently running miniprocesses within a process. Useful in programs that have multiple activities going on at once. Decomposing the application into multiple sequential threads that run in quasi-parallel, the programming model becomes simpler. Threads can share address space and the data, multiple processes cannot do this. User-space threads-Can be implimented on an OS that does not support threads since the OS thinks it is managing single-threaded processes. Processes can have their own scheduling algorithms. Kernel threads-The kernel has a thread table that keeps track of all threads, so there is no need for each process to have a thread table. Threads can make a kernel call to create a destroy threads. Kernels allow system calls that might block. Pop-up thread - When a message arrives create a new thread to handle it, rather than having an existing thread waiting. Single threaded to multithreaded conversion - Threads might try to access the same global variable, this causes a value stored by a thread in a global variable to be lost when another thread writes to that global variable. Library procedures are not designed to handle a second call while a previous call is not finished. It is also difficult to manage what thread should catch signals such as keyboard input. Also the kernel is not aware of each thread having its own stack, so the kernel cannot grow these stacks automatically like it does when it only has to deal with a single stack. Interprocess communcation issues- How one process can pass information to another, making sure two or more processes do not get in each others way(processes trying to grab the same resources at the same time), sequencing when one process requires completion of another process before it can begin. Race condition-Two or more processes are reading or writing some shared data and the final result depends on who runs precisely when. Critical region-The part of a program where shared memory is accessed. Busy wait - Continuously testing a variable until some value appears. Spin lock - A lock that uses busy waiting. Disabling interrupts-Gives user processes the ability to turn off interrupts. If a user process turns off an interrupt it will never turn on again. Software-only lock variables - Can cause two processes to access their critical regions at the same time. Peterson's solution - Each process calls enter\_region with its process number, this causes the process to wait until it is safe to enter. After that it calls leave\_region, allowing other processes to enter. TSL- Reads the contents of the memory word lock into register RX and then stores a nonzero value at the memory address lock. No other processor can access the memory word until the instruction is finished. Priority inversion problem - A high priority process becomes ready while a low priority process is in its critical region, L doesn't get a chance to run because H is busy waiting to enter the critical region, which L is currently in. Producer-consumer problem - Producer goes to sleep while waiting for the consumer to remove items from the full buffer, or consumer goes to sleep waiting for the producer to put items in the empty buffer, this leads to race conditions. Semaphore - variable type, 0 means sleep, greater than zero means number of wake ups. Binary semaphore - If each process does a down just before entering its critical region and an up just before leaving it, this makes sure no other processes try to enter the critical region. Mutex actions - A variable that represents either locked or unlocked. When a thread needs to access the critical region it sets the mutex to locked, when it leaves it unlocks it. Monitors - A collection of procedures, variables, and data structures grouped together in a module or package. Processes can call the procedures in a monitor but they cannot access the monitors internal data structures with procedures declared outside the monitor. Only one process at a time can access the monitor, if another process tries to access it the process will be put on hold until the process currently in the monitor is done. Condition variable - When wait is performed on a condition variable it causes the calling process to block. It also allows a process that was previously prohibited from entering the monitor to enter it. Message Passing - Consumer sends empty messages to the producer, the producer takes an empty message and sends back a full one. This means the total number of messages stays the same and the amount of memory needed to store those messages can be known in advance. Barrier-In phased process programs all processes must complete the current phase before any process can move to the next, a barrier stops processes from moving onto the next phase until all processes have reached the barrier. Multiprogramming - When a process is doing I/O another process starts using the CPU, this keeps the CPU busy. Spooling - Data is placed in a temporary working area so that another program can access it later for processing. DMA(direct memory access) - A chip that can control the flow of bits between memory and some controller without constant CPU intervention. Pipe - A sort of pseudofile that can be used to connect two processes. Virtual machine - Treats the OS kernel and hardware as if they were all hardware. Exokernels partition the resources among the users rather than cloning the actual machine. Microkernel - Split the operating system up into small well defined modules, only one of which, the microkernel, runs in kernel mode and the rest run as relatively powerless user processes. Mount - mounts a file system. Strict alternation - It causes extra wait sometimes because it may be a processes turn to enter the critical region but that process is working outside the critical region while another process that needs to enter the critical region has to wait. Caching is used for data that is accessed most frequently because cache memory is faster than RAM. Shell - When a valid command is typed into the shell the shell creates a child process and runs the program corresponding to the command, when the child process terminates the shell displays the command prompt again. Process control block/process table - One entry per process, contains information about the processes state such as its program counter, stack pointer, memory allocation, status of its open files, its accounting and scheduling information and other information that must be saved when the process is switched from running to ready or blocked state so that it can be restarted later as if it had never been stopped.