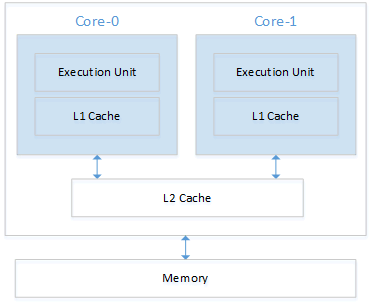
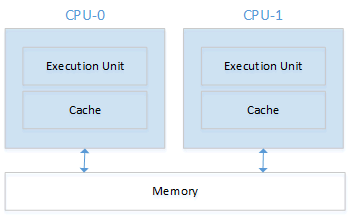
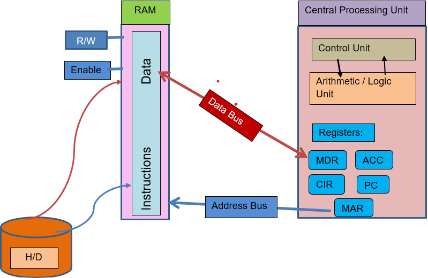
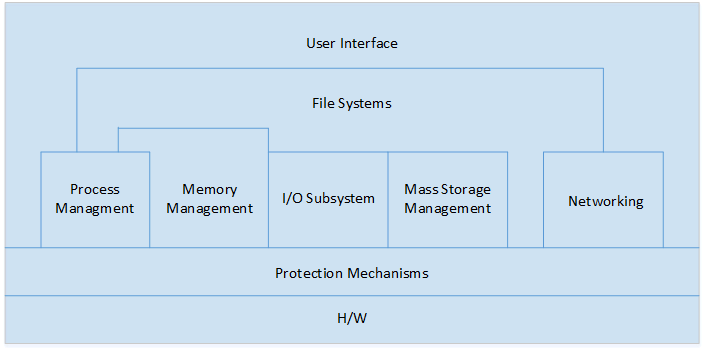
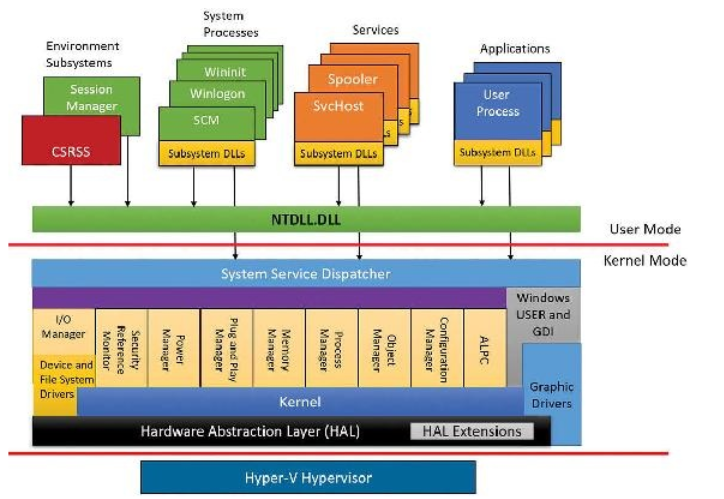
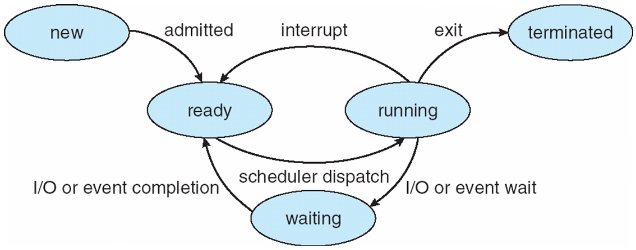
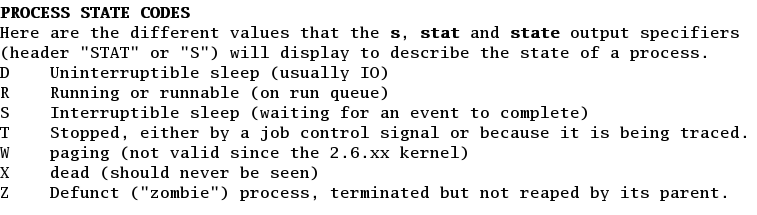
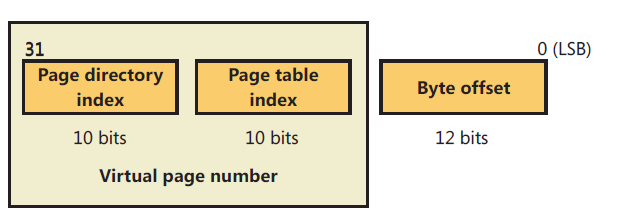
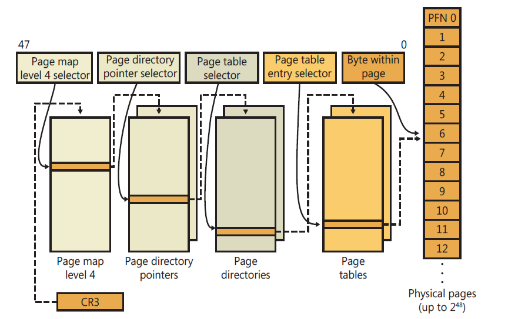
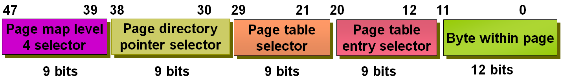
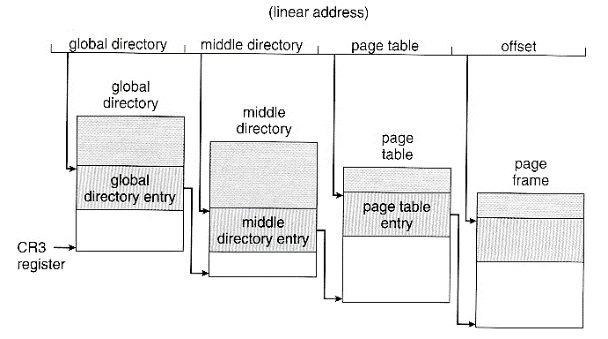
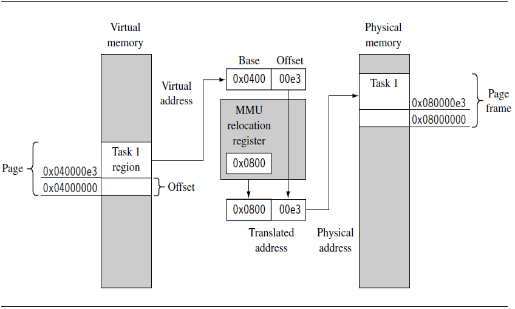
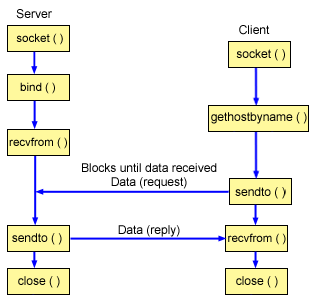
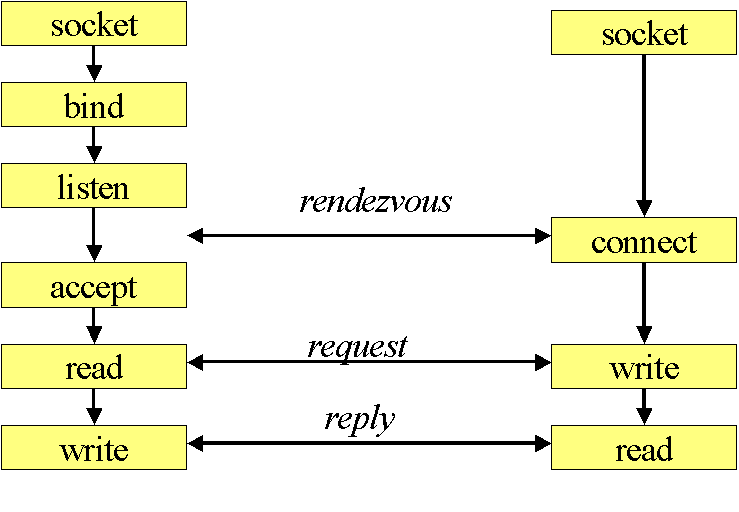
*********MODULE 1:*** OS is a resource allocator – manages all system resources like CPU time, mem space, file storage space, I/O devices. It is also a control program which controls executions of programs to prevent errors and improper use of the system. OS is the kernel required to run the system – a program running at all times. Batch jobs – similar needs running thru the computer together as a group. Good for large jobs with little interaction. Multiprogramming – efficiency – organizes jobs (code and data. -Subset of total jobs in system kept in mem. Attempts to increase CPU utilization by always having something for the CPU to execute. Timesharing – CPU switches jobs so frequently that users can interact with each job will running **Interactive.** Response time is <1 second. Each user has one program executing in mem. Process several jobs waiting to run = scheduling, Process don’t fit in mem. = swapping. Virt. Mem. Allows execution of process not completely in mem. Real-Time Systems – Hard and soft, immediate responses. **Hard** real time for medical systems, NASA, etc. **Soft** allows for delays – video games, VR, etc. Bus – transfers data between components inside a computer. CPU to Memory. Memory is an array of bytes represented in hexadecimal. HD holds the virtual mem. CPU contains registers and cache (L1 -> L2 -> L3). Cache is often used by pages volatile. RAM – volatile any executable is a process. Process gets loaded into RAM from HD. If pages are not needed they are stored in VM. ROM – non-volatile read only memory includes boot instructions. CPU can hold generic and specific registers. Stack pointer stores the head of the stack in the current process. Link Register – return address -> where the core puts the ret. Add. When it calls a subroutine. Program Counter – Addr. Of next inst. To be fetched. CPU will fetch and load from RAM. *If a computer has two cores it then has 4 logical computers (double*). Thread == Logical Computers. *Multiprocessing != Multiprogramming.* Asymmetric Multiprocessing – master/slave. Master controls all the work to be done. Symmetric (SMP) – peers, any task can be done by the CPU. \*\*Multicore uses L1 and L2 Cache\*\* Blade Servers – has its own processor and OS to operate independently. Process Mgmt. – a program in execution. Process mgmt. can create and delete, schedule processes, suspend & resume, provide mechanism for process synchronization, and synchronize inter-process communication. Memory Mgmt. – reads instructions from main mem during instruction – fetch. Allocating and deallocating mem. Space. File Sys. Mgmt. – organized into directions, access control, etc. Storage – one set of mem. For all program and data. Usually from main mem. I/O – Competes for the system bus to access memory -> device controllers -> device drivers. Networking – Software (app, Pres, Session), OS (Trans, Netw, Datalink). \*\*\*RAM is slow compared to CPU. CPU is slower than cache\*\*\* Interrupt Vector – interrupt transfers control to the interrupt service routine, saves adder. Of the interrupted instruction. Trap or Exception – software generated interrupt caused by error or user request. OS is interrupt driven. Interrupt can be triggered by software or hardware. *Mobile devices contain fewer cores and less storage than a traditional laptop.* Virtualization allows OS to run simultaneously on the same machine. **Emulation is simulation of hardware**. VMM runs natively and runs guest operating systems. Clustered System – set of multiple computer systems or nodes that are connected by a local area network. Firmware interfaces – UEFI, EFI, BIOS. On start-up, **bootstrap** program is loaded. *Firmware is hardware specific code*, stored in ROM, EPROM, Flash mem. UEFI replaced BIOS. BIOS, slow, uses **MBR** -> 32 bits for logical address block and data in 512 bytes. MBR max disk size of 2.2 TB, max 4 partitions. UEFI, boot from disk larger than 2 TB, contains data tables with GUID partition table (GPT) partition structure instead of MBR. Has 64bit Values, handles 9.4 ZB. Can do backwards compatibility if secure boot is off. **Boot Process: 1.** Power On **2.** POST -> Power on self-test (tests every element of the mother board). **3.** BIOS boot code, from hardware. **4.** MBR – tells from which OS you are booting from (very small and limited). **5.** Boot Loader – list of OS to choose from (Bootmgr – Win, GRUB – Linux) **6.** Loader loads Kernel into RAM (Winloade.exe – BIOS, winload.efi – UEFI) – Windows Kernel – NTOSkrnl.exe, Linux – vmlinuz. **7.** Kernel initializes processes, drivers, registers, etc. (NTOSkrnl initializes executive subsystem and the first process – smss,exe). **8.** Smss.exe creates a session that allows the user to work with / linux is init or system which runs rcd (scripts) – Loads kernel drivers \*win32k.sys\* then initializes crss and part of the **registry** and wininit.exe. **9.** Wininit.exe starts services like SCM, LSASS, LSM. **10.** Winlogon.exe initializes logon UI. **11.** Services.exe. To edit the registry – regedit. To modify boot – msconfig. To use WMIC – it is advanced, so you must know what you’re doing. ***MODULE 2:*** Monolithic kernel – one large program (If you make changes at all it can affect the kernel). Microkernel – Most OS run separately. They communicate via message-passing. Separate from the file system. Performance can be affected due to increased system-function overhead. Hybrid – Windows and Linux. OS protects kernel by running in 2 modes – **User and Kernel**. User -> services, accounting, I/O operation, etc. Kernel -> can execute all machine instructions, can reference all memory locations. *Layered operating system, Kernel is layer 0, Applications are ring 3*. CLI = Linux – echo $PATH, Windows is WMIC. Sbin and system32 hold executables. Linux - /usr/src contains kernel source. Hyper-V – hardware virtulatization for running separate virtual machines. Has two implementations, normal – VTLO 0 and Secure – VTL 1. Both have a kernel mode and user mode. **Security at the OS level –** *Authentication, Authorization, Audit.* System Calls – routines written in high level language eg. printf() -> write() API – specifies a set pf functions including parameters that are available to an application programmer. INVOKES SYSTEM CALLS. Win32, POSIX, Java. System calls – not in X86 user mode -> X64, For X80 -> x86. Parameters to System Calls – can be passed in registers and stack, cannot be passed to system call via API function. Command interpreter – to get and execute next user-specified command. **Compliation Process**: Preprocessor (Source Code -E), Compiler (Expands to assembly), Assembler(assemble source code to object code -s), Linker (Object code to executable code .o), Executable. **Shell is used to execute a program. Strace –** traces system calls and signals. **GDB display for next instruction – x/i $pc**. Libraries – Android – webkit, SQLite, multimedia. Dynamically Linked Libraries are only loaded when a proces is ran. 3 methods to pass parameters to the OS. 1. Pass the parameters in registers. 2. Parameters stored in a block, or table in memory, and address of block passed as a parameter in a register (Linux). 3. Parameters pushed onto the stack then popped off by the OS.  **System Calls:**  fork () – parent clones itself creating a child (-1 Error, 0 Child Procces, >0 pid of child to parent). wait() waits for the child process status. Clone(), execvp () duplicates actions of the shell. Exit() – child returns a value to its parent, getpid(), getppid(). *\*\* If a parent exits unexpectedly the process may become an orphan which is adopted by init or systemd \*\*.* System Calls for files: open() = fd, creat() = fd, read(), write(), close().  **Shell Program that interprets commands:** 1. Reads the input from STDIN. 2. Parses the line into token to the command and arguments. 3. Shell creates a new process by calling fork(). 4. The program is loaded into the new process, and the arguments are passed to the program. 5. The child process executes the program by invoking exec() system call to execute the program with the respective arguments. 6. The parent process waits for child process status. Process – a program in execution. A structure and memory space. When it’s created, it’ll request resources (CPU Time & registers, memory space, environment, signals, files, etc.) ***MODULE 3:*** Each process has its own data space, file descriptors, unique PID, owner, size, etc. Threads are a function or a task within a process that can run independently, run simultaneously, and share resources. Threads -> every process starts as a single thread – Code/Data/Files are shared, but they all have their own stack and registers. Process Control Block (PCB) or /PROC shows processes and data structures. **PCB’s are a doubly linked list** – has a pointer to the previous PCB and to the next PCB. *E\_PROCESS = Executive which points to the KPROCESS(Kernel) -> Windows*. PCB contains process state, program counter, CPU registers, CPU scheduling information, memory management, accounting time, open file, I/O status, pointer to TCB. \*\* IN USER MODE, THREADS HAVE THEIR OWN STACK, IN KERNEL MODE – THREADS ARE IN THE SAME PLACE \*\*. Context-Switch -> Stores the saved stat of the old process and load the saved state for the new process. **Process States**: new – process being created. Running – instructions are being executed in CPU. Waiting – the process is waiting for some event to occur such as I/O. Ready – the process is waiting in memory to be assigned to a processor. Terminated – the process has finished execution.



Signals – messages send to processes to notify it of an event. **SIGINT** – Kill -2, **SIGSTOP** – Kill -19 (core dump file), **SIGQUIT** – Kill -3, **SIGKILL** – Kill -9. SigStop cannot be ignored, but can also be resumed – Traced State. SigKill terminates immediately, cannot be ignored. Can cause damage. Kill -pid default is SIGTERM (-15) this terminates and releases resources properly and deallocates memory. To **terminate a process:** *Normal Exit – Voluntary, Error Exit – Voluntary, Fatal Error – Involuntary, Killed by another process – involuntary.*

A close up of a map

Description automatically generated**Windows Process Creation CreateProcess (function): 1.** Validate parameters; parse, validate, and convert the attribute list to its native counterpart. **2.** Open the image file (.exe) to be executed inside the process. **3.** Create the Windows execuite process object EPROCESS. **4.** Create the initial thread (stack, context, and Windows executive thread object). **5.** Start execution of initial thread **6.** Complete initialization of address space, load DLL’s and begin execution. **Signals Programming:** a process can receive a signal or man signals at a time. The three are: Accepts the default action: accepts the signal sent. Ignores the signal: Ignores the signal sent. Catch signal and change the behavior. Inter-process Communication (IPC) – processes may be independent or cooperating. Processes communicate within the same system for reasons like: information sharing, computation speedup, modularity, convenience. IPC uses shared memory – faster than message passing and message passing – a good technique for small message. Pipes – two processes to communicate. Unidirectional. Require parent-child relationship. Lets you write too and read from. Socket Types – Stream sockets (TCP) and Datagram Sockets (UDP). **In Layer 4 – a port number is required.** *Linux uses signals to inform a process of asynchronouse events and to handle exceptions.* ***MODULE 4:*** Memory is a linear array of bytes represented in Hex. The CPU fetches instruction from memory according to the value at the program counter. Each process is divided into pages. Each page is 4096 = 2^12 = 4Kb. Eg 20 mb of memory / 4kb = 5000 pages. **MMU converts the logical address to physical address.**  Page Table holds the page number and corresponding frame number and its validty (or not). Stall: Lag between CPU & memory. It Wastes cycles. **Memory Management Functions ->** System services for allocating, deallocating, and managing virtual memory. Access to a region of memory is set as read-write, read-only, or no access. **Base Limit & Registers:** Base = contains base address of a memory region. Limit = contains the region size. **Base + Limit = Max Allowance.** Binding Instructions and Data to Memory – address binding of instructions and data to memory address can happen at three different stages: **Compile Time:** If memory location known prior, absolute code can be generated; must recompile code if starting location changes. **Load Time:** Must genereate relocatable code if memory location is not known until compile time. **Execution time:** Binding delayted until run time. Needs hardware to support for address maps. **Stack:** Temporararily stores – local variables, parameters, return address of a function. Stack grows toward lower address. LIFO. SP points to the top of the stack. Can have stack overflow if it isn’t large enough. **Heap:** malloc alloates bytes and returns a pointer to the allocated memory. Brk() and sbrk() calls change data segment. Memory leak can be generate if malloc is not managed properly. **Memory Allocation:** dynamically partitioned according to process sizes as each process is loaded. 🡪 Higher overhead but allows flexibility 🡪 Involves tracking used and available memory 🡪 Process can be loaded if there’s a big enough hole (contiguous block of memory). **External Fragmentation 🡪** Heap: When memory is unused but too fragmented to be used as contiguous space. Can be solved by compation. **Dynamic Storage Allocation:** 1. First-Fit: first hole in memory that is big enough for the process. 2. Best-Fit: Smallest hole found in memory that can fit the process (no waste). 3. Worst-Fit: Biggest hole found in memory that can fit the process (Lots of waste). **Cache:**  Write-Back: cache controller writes data modifications to cache and doesn’t update RAM until necessary. Write-buffer or write through: cache controller writes data to RAM and cache simultaneously. ***Dirty Bit:*** A page can be written to storage. **Virtual (Logic):** assigned by the computer & linker when locating in memory – contains page number and offset. **Physical:** where the actual H/W components of main memory where the programs are physically located. \*\*They differ in execution time\*\*. Processor determines page size and OS. **E.g. 21 MB / 4 KB = 5.5K or 5500 pages** Whatever isn’t used is considered internal fragmentation. **MMU:** Hardware based memory management unit that translates virtual address to physical. \*\*ARM MMU HAS MULTIPLE RELOCATION REGISTERS\*\* **Paging:** a single relocation register can only translate a single area of memory. **Page Frame:** phsyical memory pointed to but the translation process. **Page Table:** contains descriptions of virtual page information organized by birtual address and the page frame in virtual memory. *Page Directory:* index poinint to each page table. **Structure:** 10 bits wide, allowing you to reference up to 1024 4-bytes PTE’s. *To calculate - 4 GB / Virtual Memory* = Page Tables. **X64 Virtual addressing –** 256 TB of virtual space, onlyy 48 bits implemeneted. Page sizes of 4kb, 2Mb, 1 GB. Four level paging. Only allows for 16EB. **TLB** stores recently used page translations. 🡪TLB Hit – provides translation of physical address. 🡪 TLB Miss goes to MMY by searching page tables. \*\*MMU translates the first bits – the base address \*\*. **Address Space Layout Randomization** – Randomizes where information is being stored.