Operating Systems use what are known as processes and they are programming in execution. These act as a container that holds all the information needed to run a program. Interprocess communication is when two or more processes must communicate in order to get a job done or synchronization. Alarm signals cause the processes to temporarily suspend whatever it is doing, save its registers on the stack, and start running a special signal-handling procedure. Directories are used to group files together and branch off of a root folder. Each process has a current working directory and any time a file is opened its permissions are checked. Mounting system calls allows for the file system on a cd-rom to be attached to the root file system wherever the program wants it to be. Piping is something that can be used to connect two processes. We use rwx to denote if a file has read, write, and access permissions and is broken into three groups, owner, owner's group, and everyone else. Example: rwx r-x--x means owner can read, write or execute the file r-x means group members can read or execute but not write, and - - x indicates a search permission for everyone else. Only system calls enter the kernel, procedures do not. TRAP instruction: user space points to a caller that it will use to execute a system call, once that system call is being run we are trapped into the kernel until that system call is over by returning some value to the caller. Operating System structure: Monolithic systems run on a single program kernel, crash in any procedure takes down the entire system. Basic structure for an operating system: 1) main program invokes the requested service procedure. 2) A set of service procedures carry out system calls. 3) a set of utility procedures that help the service procedures. Layered Systems: organize the operating system hierarchy, each one is constructed upon the one below: Layer 5: the operator, Layer 4: user programs, Layer 3: input/output management, Layer 2: operator-process communication, Layer 1: memory and drum management, Layer 0: processor allocation and multiprogramming. Microkernels: achieve high reliability by splitting the operating system up into small, well defined modules, only one which the microkernel runs in kernel mode and the rest run as relatively powerless ordinary processes. Client server model: servers provide some service and clients use those services, this is done by utilizing some network and messages are exchanged back and forth for the appropriate requested service to be obtained. Virtual machines: exact copies of bare hardware, including kernel/user mode, input/output, interrupts, and everything else a real machine has. Type 1 hypervisor: for end users who want to be able to run two or more operating systems at the same time. Type 2 hypervisor: make use of the host operating system whereas type 1 must do all the managing on their own. Paravirtualization: approach of handling control instructions is to modify the operating system to remove them. Exokernels: Job is to allocate resources to a virtual machine and then check attempts to use them to make sure no machine is trying to use somebody else's resources.

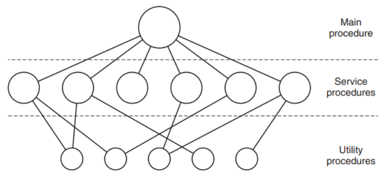
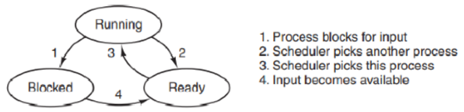


Figure 1-24. A simple structuring model for a monolithic system.

Multiprogramming refers to the switching of multiple processes back and forth and is the execution of multiple processes at a particular time.. There are four events that cause a process to be created: 1) System initialization (power on) 2) execution of a process (creation system call by running a process) 3) A user request to create a new process 4) initiation of a batch job. Processes terminate for the following reasons: 1) normal exit (voluntary) 2) error exit (voluntary) 3) fatal error (involuntary) 4) killed by another process (involuntary). The difference between a process and a thread is that a process is a program in execution and its threads are sub features of the process. A good example of this would be a webpage running and some application like betterttv running as a feature of the main process. Process States: 1) running (using the GPU), 2) ready (runnable), 3) blocked (unable to run).



Running
Blocked
Ready
1
2
3
4
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Processes are placed into a table where the scheduler will decide which processes are going to be handled. Process tables include: state, program counter, stack pointer, memory allocation, status of its open files, account and scheduling. Threads have the ability for the parallel entities to share an address space and all of its data among themselves. Threads are easier to maintain than a process because we can destroy them easier, speeds up computing by letting activities overlap. Threads offer parallelism and block system calls, Single threaded processes offer no parallelism and block system calls, FSMs offer parallelism, do not block system calls, use interrupts. Thread model: processes group related resources together, threads belong to a process and execute tasks. Processes are used to group resources together; threads are the entities scheduled for execution on the CPU. User Space threads: Each process needs its own thread table, and threads that are implemented in the user space are faster than trapping into the kernel. Some issues with user threads: blocking system call implementation, if a thread starts running, no other thread will process unless the first voluntarily gives up CPU time. Kernel threads: No run time system is needed, no thread table in each process, all calls that might block a thread are implemented as system calls, sometimes threads are recycled. Issues with Kernel threads: signals, unsure of which thread will handle the signal. A race condition is when two processes attempt to read or write some shared data and the final result depends on who runs precisely when. Four conditions to hold: 1) no two processes may be simultaneously inside their critical regions, 2) No assumptions may be made about speeds or the number of CPUs, 3) No process running outside its critical region may block any process, 3) No process should have to wait forever to enter its critical region. Mutual exclusion solutions: Lock variables - allows a lock to switch from 0 to 1 to determine if a critical region is accessible, but this method may override data. Strict Alternation - two loops are constantly checking the status of the critical region, 0 or 1, accessible or not accessible respectively. Peterson's Solution - uses an array to demonstrate which process is interested in entering a critical region. TSL Instruction: Test and set lock, which locks the memory bus to prohibit other CPUs from accessing memory until finished and can be turned to 1 by any process that uses a TSL instruction. XCHG may also be used which exchanges contents of two locations automatically. Sleeping and Wake Up - causes caller to block, until another process wakes up. Semaphores are used to control the state of processes and determine which can access the critical region. Semaphores have two states, either up or down, down meaning the value is zero and the process is put to sleep, up increments the semaphore and one the the processes is chosen by the system and is allowed to complete its down. Semaphores are an abstract data type used to access common resources by multiple threads and avoid critical region problems in concurrent systems like multitasking in operating systems. It is important that we keep critical regions small. Synchronization: allows for the processes to properly trade off, for example when the producer stops when the buffer is full, and the consumer when the buffer is empty. Mutexes: can be in a state of locked or unlocked and used for accessing a critical region. Futexes: Fast user space mutex, which avoids dropping into a kernel unless it has to. Condition variables: allow a thread to block due to some condition not being met, differ from semaphores because they do not have memory, meaning that if a signal is sent to a condition variable on which no thread is waiting the signal is lost. Monitors: Collection of procedures, variables, and data structures that are all grouped together in a special kind of module or package. Processes outside of the monitor can not access the internal variable of the monitor but can call procedures. Message passing: Uses send and receive, which like semaphores and unlike monitors, are system calls rather than language constructs. Barriers: Some applications divide into phases and have the rule that no process may proceed into the next phase until all processes are ready. Scheduling decides which process will go next when two or more processes are in a ready state using an algorithm. Scheduling algorithms are classified as Nonpreemptive, which picks a process to run and then just lets it run until it blocks or voluntarily releases CPU, and preemptive, which picks a process and lets it run for a maximum quantum (fixed time). 3 types of scheduling: 1) Batch: no users impatiently waiting at terminals for a quick response to a short request, and both preemptive and nonpreemptive work, essentially a lot of tasks are queued and run in a large batch. 2) Interactive: preemption is essential to keep one process from hogging CPU and denying service to others, this is how modern computing is done and how gaming consoles typically work. 3) Real time: preemption but sometimes not needed because the processes know that they may not run for long periods of time and usually do their work and block quickly, like a traffic light, real time systems are dependent on time and must have processes stop and start on a rigid time basis. Scheduling algorithms take 3 things into consideration: 1) fairness, 2) policy enforcement, 3) balance. Batch systems want throughput, turnaround time, and CPU utilization. Interactive systems want response time, and proportionality (meeting a users' expectations). Real-time systems want to meet rigid deadlines, and have predictability. Batch systems algorithms: First Come First Served: process is assigned the CPU in the order they have been requested (queue). Shortest Job First: prioritizes the smaller jobs to get CPU time first and then larger processes get placed at the end of the queue. Shortest Remaining Time Next: when a new job arrives, its total time is compared to the current process' remaining time. Scheduling in Interactive Systems: Round Robin: Each process is assigned a time interval, known as a quantum, during which it is allowed to run, if it is still running toward the end of the quantum, the CPU is preempted and given to another process. Setting the quantum too short means too many switches and lower cpu utilization, too long may cause poor response time for requests. Priority Scheduling: Takes external factors into account, processes are assigned a priority, and the runnable process with the highest is allowed to run, process switches occur when a process may be favored too heavily or has had too much run time. Shortest Process Next: We use the aging technique to estimate runtime based on past behaviors, this technique of estimating the next value in a series by taking the weighted average of the current measure value and the previous estimate. Guaranteed Scheduling: processes are given 1/n of CPU cycles for all n processes, comparing the ratios the lowest is given CPU time until the ratio is just above its closest process. Lottery Scheduling: processes are given tickets, and a ticket number is pulled and that process is run, some processes may be given more tickets so that they are considered a higher priority. Fair Share Scheduling: The number of users is taken into account and the process is split amongst them, if there are 2 each would get 50%. Scheduling Real-Time Systems: schedulability is considered possible if a periodic m event occurs i times with a period of pi that requires ci seconds of cpu time: $\sum_{i-1}^{m} \frac{c_i}{p_i} \leq 1$ and can either be static or dynamic, meaning their scheduling decisions happen before the system starts running or during runtime, respectively. Policy Versus Mechanism: separating the scheduling mechanism for the scheduling policy allows the scheduling algorithm to be parameterized in some way, but the parameters can be filled in by user processes. Thread scheduling: threads switch with user-level threads and take a handful of instructions or kernel level threads require a full context switch changing the memory map and invalidating the cache but block on I/O does not suspend the entire process as it does in user-level threads. Here the kernel picks either a process entirely or an individual thread. Example: Process A has three threads A1, A2, A3, and Process B has B1, B2, B3, if we use process A we have a possible A1, A2, A3..An, but not A1, B1, A2, B2, if we use threads we can have them execute how we want. Starvation: all programs continue to run indefinitely but fail to make any progress.

The main functions of an operating system are to provide a programmer with a clean abstract set of resources instead of messy hardware, and managing those resources. Time sharing is where a central computer is used and is connected to many different terminals, all the requests from different terminals are sent to this central computer and carried out, whereas multiprogramming takes place on a machine by one person but is utilizing many processes within that machine simultaneously. The difference between kernel mode and user mode is that kernel mode allows for an individual to have full access to hardware and its instruction set, whereas user mode maintains protections by permitting users to do limited operations. Decoupling a processes address space from the machine's physical memory allows for the kernel to be smaller and crashes can be managed easier.