* Operating System – Body of software responsible for making programs easy to run
* Virtualization – Take physical resource and turn it into more accessible resource
* Virtual Machine – Another reference to an operating system
* System Calls – Allows the os to run programs, access memory and devices
* Resource Manager – Shows various CPU and Memory usage
* Concurrency – Work on multiple things at once. Why??
* Persistence – Allows things to be stored when there isn’t any power, where ram loses memory when power is lost.
* Design Goals – Design of the system to bring all of these things together
* ***Process*** – A running program
* Time Sharing – Allow a resource to be shared
* Context Switch – Allow the CPU to stop running one process and start another
* Stack Pointer -
* Frame pointer - The easiest way to understand is to think about method calls. Frame pointer + 0 = where we came from. Think of recursion. An address in the stack. Tells us where we are
* Program Counter – which instruction of the program is currently being executed
* Process Creation
* Eagerly vs Lazily – Eagerly loads pieces of code that are used often. Lazily loads code only as it’s needed
* Process States –
  + Running – running and executing instructions
  + Ready – ready to run, but the OS has chosen not to run it yet
  + Blocked – Dependent on something else before it can run
* Process List – Simple data structure that allows OS to run multiple programs at once
* Zombie State – A program that is done running but hasn’t closed yet. The child process is running while the parent has exited.
* ***Fork*** – Create a new process **Wait** – Wait for your processes to sync up before continuing **Exec** – Calls another process
* ***Scheduler*** - We have a number of different things that take care of other things. Manages the processor. What will be running and what won’t be running.
* Workload – Processes running in the system **Jobs** – A process
* Turnaround Time – time at which the job completes minus the time at which the job arrived in the system. How long the process ran from open to close, not important because it’s user dependent
* Fifo – First in first out **Convoy Effect** – When a large process ends up holding up a bunch of smaller processes
* SJF - Shortest Job First , good in a perfect world. But we don’t live in a perfect world, so convoy effect still takes place. Good for turnaround time
* STCF – Shortest Time-to-Completion First, a preemptive scheduler (preemptive shortest job first) Good for response time
* Preemptive – Stop a running process and start another while going back to original **Non-preemptive**
* Response Time – Time a job arrives in the system to the first time it is scheduled (first run – arrival)
* Round Robin – Rather than running a job to completion, it runs each process for a specified amount of time in a circle until finished
* Time Slice – must be an interval of the timer-interrupt
* Scheduling quantum – same things as time slice
* Oracle Algorithm – The scheduler knows the length of each job. Importance because it gives you the best case scenario. You cannot do better than the algorithm with the given constraints.
* ***MLFQ*** – Optimize turnaround time (run shorter jobs first). The os doesn’t know how long a process is going to be though. Also minimizes response time. Uses the past to predict the future.
* Turing award – Highest award in CS. Awarded to Corbato
* Priority Queue – Used to rank processes on which ones should be completed first
* Linux Scheduler (completely fair scheduler) – Used in Linux systems. The process with the lowest execution time is ran for a minimum of the maximum execution time. If it’s complete it drops off, if not it’s inserted back into the list with how much time is left to run it.
* Red-black tree – Self balancing binary search tree
* ***Address space*** – An abstraction of physical memory used by the OS with Users in mind.
* Sharing memory – Allows for memory to be divided up into sections and utilized by multiple programs and processes
* Multiprogramming – Multiple processes running at the same time. The OS knows when to switch between them
* Utilization – Multiprogramming increases the utilization of the CPU by making it be used more often
* Efficiency – Increased utilization equals increased efficiency which was important back when computers cost a fortune
* Time Sharing – Allows for a program to run for a period of time, and the process be saved to a disk and ran at a later time
* Interactivity – Many people using the system waiting for return data
* Protection – Making it so a process can’t read or write to another processes memory
* Easy to use - End user kept in mind
* Code - Instructions
* Stack – used to keep track of where it is in the function call chain as well as to allocate local variables and pass parameters and return values to and from routines
* Heap – Used for dynamically allocated, user-managed memory, such as that you might receive from a call to malloc() in C or new in an OOL.
* Virtual Address – When a program tries to put something into memory [0] the hardware and software don’t actually put it there
* Transparency – Make it so the program running isn’t aware that it’s using virtual memory
* Efficiency – Make virtual memory as efficient as possible in terms of time and space
* Protection – Protect processes from one another including the OS itself
* Isolation – Each program is in it’s on cocoon isolated from potential malicious software
* ***Stack*** ­– Automatic memory, managed implicitly by the compiler
* Heap – Used for memory that is needed after the end of the function that called it. All allocation and deallocations are handled explicitly by the user
* Malloc – you pass the size of something asking for room on the heap. Needs #include <stdlib.h>
* Free – Frees memory that was allocated by the malloc system call
* Segmentation fault – using memory that hasn’t been allocated
* Memory leak – forgetting to unallocated memory that had been declared
* Brk Break – Used to change the location of the end of the heap. Used to increase or decrease the size of the heap depending on the address passed
* Mmap – Creates an anonymous memory region that is used with swap space. After declared it’s treated like the heap
* ***Base and Bounds*** – (Dynamic relocation) base register and bounds register, this pair allows us to place the address space anywhere we’d like in physical memory, and do so while ensuring that the process can only access its own address space
* Translated address – Physical Address = virtual address + base (where the OS loads the program)
* Virtual address – each memory reference by the process is a virtual address
* Physical Address – is issued to the memory system via the formula above
* MMU – Part of processor that helps with the address translation and holds the base and bounds register
* Kernel Mode – Privileged mode, where it can access the entire machine
* User Mode – Limited to what they can do
* Exception Handler – Where the CPU stops a bad process and returns
* ***Segmentation*** – Allows us to support a large address space with a lot of free space between the stack and the heap. Instead of one base and bounds pair in the MMU, have a pair per logical segment of address space
* Segment – A contiguous portion of the address space of a particular length
* Share bit
* Protection bit – A few bits per segment indicating whether or not a program can read or write a segment
* Segment table – Required to support many segments. Allows system to be more flexible
* External fragmentation – Little holes of free space in physical memory due to various segments of different sizes
* Internal Fragmentation - Internal fragmentation is the wasted space within each allocated block because of rounding up from the actual requested allocation to the allocation granularity. External fragmentation is the various free spaced holes that are generated in either your memory or disk space. External fragmented blocks are available for allocation, but may be too small to be of any use.
* ***Free-space management –*** Easy with fixed variable sizes, harder with dynamic variable sizes
* Paging -
* Splitting - Since allocated space might be smaller than free space, we might want to split the block. Splitting will find a free chunk of memory that can satisfy the request and split it into two. The first chunk it will return to the caller; the second chunk will remain on the list.
* Coalescing - Join (coalesce) with next/previous blocks, if they are free Coalescing with next block
* Best fit – Search through the free list and find chunks of free memory that are as big or bigger that the requested size. Then, return the one that is the smallest in that group of candidates.
* Worst fit – Find the largest chunk and return
* First fit – Finds the first block that is big enough
* Next fit – Keeps a pointer where it was last looking, spreads the searches for free space throughout the list more uniformly avoiding splintering of the beginning of the list
* Binary buddy allocation – Used to make coalescing simple. Free memory is first conceptually thought of as one big space of size 2^n. When a request for memory is made, the search for free space recursively divides free space by two until a block that big enough to accommodate the request is found.
* ***Paging -***Chop up space into fixed-sized pieces. Instead of splitting up a process’s address space into some number of variable-sized logical segments, we divide it into fixed-sized units, each of which is a page
* Page
* Page Table – Keeps track of where pages are in physical memory. This is a per-process data structure.
* Tlb - Translation lookaside buffer. Changes a virtual address into a physical address of where it actually is. Take pages and tables into account. What is it, what’s its purpose, why do we do it? It’s a part of the chips MMU and is a hardware cache of popular virtual to physical address translations. By providing a small, dedicated on-chip TLB as an address -translation cache, most memory references will hopefully be handled without having to access the page table in main memory. This the performance of the program will be better.
* ***Swap -***Space reserved on the disk for moving pages back and forth OS can read and write from here.
* Page-fault – The act of accessing a page that is not in physical memory
* Page-in – To move a page from swp to memory
* Page out – To move a page from memory to swp
* Page-replacement policy – The process of picking pages to kick out.
* Swap daemon – A thread that is in the background used to keep open memory based on the watermarks
* Page daemon
* Cluster -Group a bunch of pages together so they can all be written out at once
* ***Cache miss –*** To minimize the number of times that we have to fetch a page from disk
* Cache hit – the number of times a page that is accessed is found in memory
* AMAT- Average memory access time = (Page hit \* cost to access memory) + (Page miss \* Cost to access disk)
* Optimal/oracle policy – Leads to the fewest number of misses overall. Replaces the page that will be accessed furthest in the future
* Cold start/ compulsory miss – Building of the Cache
* FIFO – First in first out
* Random – Picks random page to replace under memory pressure. Simple to implement, but not very intelligent
* LRU – Least Recently Used, the more recently a page has been used, the more likely we will use it again. Replaces the least-recently used page
* LFU – Least Frequently Used, the more frequently we use a page, the more likely we will use it again. Replaces the least frequently used page
* MFU- Most frequently used (doesn’t work)
* MRU- Most recently used (doesn’t work well)
* Clock Algorithm - How does the OS employ the use bit to approximate LRU? Well, there could be a lot of ways, but with the clock algorithm [C69], one simple approach was suggested. Imagine all the pages of the system arranged in a circular list. A clock hand points to some particular page to begin with (it doesn’t really matter which). When a replacement must occur, the OS checks if the currently-pointed to page P has a use bit of 1 or 0. If 1, this implies that page P was recently used and thus is not a good candidate for replacement. Thus, the use bit for P set to 0 (cleared), and the clock hand is incremented to the next page (P + 1). The algorithm continues until it finds a use bit that is set to 0, implying this page has not been recently used (or, in the worst case, that all pages have been and that we have now searched through the entire set of pages, clearing all the bits).
* Dirty bit – Shows if a page has been modified or is clean
* Demand paging – The OS brings the page into memory when it is accessed, on demand, as it were
* Prefetching – The OS can guess that a page will be used and get it ahead of time
* Thrashing – If a system is out of memory, it will continue page things in and out. Linux runs an out of memory killer