**Operating System Definition**: Program that runs the hardware, runs on top of the architecture

**Storage Definition**: bit = 1/0, byte = 8 bits, kilobyte = 1024 bytes, megabyte = 1024^2, giga = 1024^3

**Architecture Definition**: the layout and connection of the hardware, Intel, power, types of machines

**Interrupt Vector**: an array of addresses/pointers to the location of routines frequently called by interrupts

**Interrupt**: a called made to the system from the hardware, then used to transfer control

**Open Source Software:** Software made available in the source code and not compiled binary file

**System Calls**: an interrupt to the OS sent by the software

**System Programs:** programs that ship with the OS and are part of the system operations

**Process and Kernel Core Dump**: a snapshot of the memory to be opened and run by a debugger, core = program, Kernel is crash

**Microkernel**: very basic, used to pass messages from one module to the next, small memory management.

Advantages: ease of extending into the OS, all new things are added to the user space which does not modify the kernel more security & reliability most processes run in user mode, if a process fails it doesn’t affect the entire OS

Disadvantages: performance decreases due to increase in system function and overhead

**Layered Kernel:**

Advantages: Has a simplicity of construction and debugging. Each layer doesn’t care about the implementation of the other layers.

Disadvantages: How to determine what is appropriate in the layers, less efficient each layer adds overhead.

**Modular Kernel:** Has modules connected and some memory management and processes in the kernel.

Advantages: Dynamic features involved in the core, more flexible that layered model, more efficient

**Boot Block:** At the top of memory, run by BIOS, contains the kernel, loads the OS for the hardware, checks machine state, in read only memory note easily affected.

**Device Management:** kernel tracking the attached devices.

**Process Creation:** Fork(), Exec(), Parent and child

**Process Termination:** zombie vs Orphan

**Process Run­time Sections:** Stack (read/write no exe), Heap (read/write no exe), Data (read only No exe), Text (Read Only, EXE)

**Process Control Block:** the way the process is represented in the OS, it includes pieces of information (process state, PID, program counter, registers memory limits, lists of open files)

**Context Switching:** from a CPU task to a kernel routine, saves information from PCB switches to kernel mode form user mode aand the loads the info back to PCB or queue

**Process Context Switching:**  1. Interrupts, 2. Time slice expired 3. Fork () or child 4. I/O request.

**Process Execution States:** new/init() process being created, Running – instructions are being executed, waiting- waiting for and event or completion, ready – wating for a processor, terminated finished execution

**Shared Memory:** when two process access the same memory locations, they communicate via the memory locations. Producer / Consumer memory segment lies with which process created it.

**Message Passing:** When the kernel passes information from one process to another, useful in a distributed environment,/ sockets / ports/ buffers / pipes / calls

**Threads:** A basic unit of CPU utilization, a program counter, thread id, register set and stack is part of a process, the process shares it text / data and OS resources, each thread performs a specific task. Benefits: responsiveness / resource sharing / economy scalability

**Multithreading Models:** many to one, many user threads to one kernel thread, done by thread library in user mode, efficient, but block by on thread blocks the system, no room for parallelism. One to one – one user thread to one kernel thread, allows concurrency and parallelism, lots of overhead because of creation of threads, most implementations limit the number of threads. Many to Many user threads draw back are minimal, no blocking and more real parallelism.

**Lightweight Process:** the data structure in between a user thread and the kernel thread

**Thread Pool:** a api for creating and managing threads, a number are created on startup and then wait for work, faster than creating new threads, limits the number of threads, strategies for doing different types of work.

**APC**(Asynchronous process control): one the parent creates the child the parent continues and parent and child run concurrently, the threads do not need to know what the other is doing. (Synchronous, when a parent waits for the child / children to run before executing: a fork / join strategy)

**Amdahl’s Law:**  performance gains by adding multiple cores: speedup <= 1 / (S + ((1-S)/ N)

**Task and Data Parallelism:** distributing subsets of the same data across multiple cores, and performing the same operation on multiple cores

**Race Conditions:** When two process are trying to access the same data concurrently and it depends on the order they are in

**Atomic Operation:**  an operation where a processor can read and write a location in the same bus operation. This prevents other operations from doing that at the same time.

**Purpose of Locks:**  to protect critical sections and

**Critical Section Problem:** 1. Mutual exclusion: if P1 is working p2 can’t and visa-versa 2. Progress, must not be in a remainder sections 3. Bounded waiting limits the number of times a process can enter the critical section.

**Peterson’s Solution:** limited to only two threads and this uses a variable and an array to show that when the Var = array[I], acess is allowed to enter the critical section.

**Counting and Binary Semaphores:** semaphore: an integer value that can only be accessed via two atomic operations: wait(), siginal(), counting semaphore() range over an unrestricted domain. Binary semaphore can be 1 or 0. These are similar to a mutex lock, counting semaphores can be used to give access to a given resource consisting of a finite number of instances.

**Priority Inversion:** must have two priorities, when a lower priority starves a higher priority because it preempts the higher priority that is running.

**Process Scheduling Algorithms:** (fcfs) process the first process or on the one with the longest wait time non-preemptive (SJF) non-preemptive faster than fcfs process the next job according to the amount of run time (how do you know)? Shortest remaining time next(preemptive) same as SJF but preemptive priority(preemptive) gives the CPU to the highest priority, round robin gives equal time to each process based on a time slice or time quantum similar to FCFS but adds preemption

**○ Multilevel Queue:** scheduling algorithm based on being an interactive process or background process, the foreground / background processes have different needs and times, with each queue having its own algorithm.

**○ Multilevel Feedback Queue:** Allows a process to move between queues, this is based off of characteristics of the cpu bursts, \*number of queues \* scheduling algorithm of each queue \* method of determining priority \* method to determine which queue has service when needed

**Preemptive:** Allowing a process to interrupt a running process. **Non-preemptive:** no ability to interrupt a running process.

**Scheduling Effectiveness Measurements:** CPU utilization, throughput number of processes per unit time, turnaround time = time from submission to completion, waiting time = amount of time spent waiting after submission to be processed, response time = amount of time between submission and first response.

**Event Latency:** time from when the event happened to when it is served. **Interrupt Latency:** time form the signal to the time it is served. **Dispatch Latency:** time to stop one process and start another.

**Starvation:** When a process has a long wait time or response time because it is preempted by other priorities or processes.

**Real­Time Scheduling:** CPU scheduling for real-time processes

**Processor Affinity:**  moving a process from memory and CPU and back and forth.

**NUMA:**  non uniform memory access, allows for storage of processes in noncontiguous locations, symmetric – each process is self-scheduling/ complex and efficient: asymmetric—one processor handles all the scheduling decision and the other processors just run user code, not complex and not efficient.

**Load Balancing:** the moving of a process from one core to another when that core is not being used.

**Deadlock Definition:** When P1 is waiting for P2 and P2 is waiting for P1. Must occur under these conditions: 1. Resources must be mutually exclusive (not shareable) 2. Must have a “hold and wait” condition 3. No ability to be preempted (only unlock my own resource) 4. Circular waiting ( each must be waiting on the other)

**Handling Deadlocks: (Most OS just ignore it)** Avoidance vs Prevention. Avoidance 1. System must know a max # of resources to allocate 2. Bankers algorithm Prevention: make one of the 4 conditions not available. 1. Make things shared / or read only 2. Every process must use multiple resources 3. Have the OS remove locks 4 enumerate resources and use them all in order.

**Resource Allocation Graph:**  P’s are processes, R’s are resources, directed edges show how the resources are allocated or requested.

**Banker’s Algorithm:** safe or unsafe, matrix of Allocated, available, max and need, can an order be found where all processes can be run at max.

**Dining Philosophers Problem:**  5 philosophers, 5 chopsticks, each philosopher must have a chopstick, remove a philosopher, asymmetric odd l then r, even r then L

**Swapping:** the process of moving from memory to storage, typically a disk, benefit of having programs larger than the size of memory

**Contiguous Memory Allocation:** allocating memory in contiguous blocks, logical address + relocation register = physical memory

**Internal Fragmentation:** A hole in the memory because the program isn’t big enough to fill up the block. **External Fragmentation:** holes in memory that add up to being big enough to run a program but not in a contiguous block:

**Relocation Register:**  Like a base register, but it is located in the MMU

**Logical Address:** An addressed used by the CPU to access memory (virtual address)**Physical Address:**  Where the process actually is in memory (an address loaded into the MMU)

**Memory Management Unit:** Maps the logical address dynamically by add the value of the relocation register

**Paging and Page Tables:** Allows the physical address space of a process to be noncontiguous, physical memory gets broken into Frames and logical section into pages

**Page Reference:** put in the TLB as a place to go find the memory

**Page Offset:** part of the address that gets added to the base to get to the physical address

**TLB:** Translation look aside buffer, which is in the hardware of the CPU that looks up the pages faster by not having to go to memory

**○ Hits and Misses:** hits are when a page is found in the TLB / Misses are not in the TLB and generate a fault to go look in memory

**○ Effective Access Time:** amount of time it takes to access physical memory address

**Demand Paging:** Used in virtual memory to load a page into memory during execution

**Copy­on­Write:** Allowing the Parent and Child process to initially share the same pages

**Page Replacement Algorithms:**  FIFO, Optimal Page Replacement, least recently used, Second Chance

**Thrashing:** replacing the pages too often and causing a lot of overhead

**Memory­Mapped Files:** placing file locations into memory for faster retrieval

**Kernel Memory Allocation**

**○ Buddy System**

**○ Slab Allocation**

**Basic Methods of File System Access**

**File Open Tables**

**○ System­wide**

**○ Per­Process**

**File Access Protection**

**File System Mounting**

**Directory Implementation**

**○ Acyclic Graph Structure**

**Relative vs. Absolute Path**

**File System Allocation Methods**

**○ Contiguous**

**○ Linked**

**○ Indexed**

**Free Space Management**

**○ Bitmapped**

**○ Linked**

**Journaling**