Process: unit of activity characterized by execution of sequence of instructions, current state & associated set of system resources, consists of program code & associated data at least
Trace: behavior of individual process by listing sequence of instructions
Dispatcher: switches processor from one process to another
Process Creation: assigns identifier, allocates space, initializes process control block, sets appropriate linkages, creates or expands other data structures
Process States: New, Ready, Blocked, Ready/Suspend, Blocked/Suspend, Running, Exit; process creation state is special (not in memory yet, code not executed yet); suspended process may not be removed until agest (process that sent command to suspend, ex: itself, parent, OS) orders removal explicitly
OS Control Structures: memory tables (allocation of main & secondary memory, protection attributes, info needed to manage virtual memory), I/O tables (manage devices & channels, has status of operation & location in main memory used as source/destination of transfer), file table (existence of files, location on secondary memory, current status, info may be maintained & used by file management system), process tables (has reference to memory, I/O & files, tables themselves subject to memory management)
Process Image: program, data, stack, PCB
Process Control Block (PCB): most important data structure in OS, for protection only handler is allowed to read/write
PCB parts: Consists of Identifiers, processor state (user visible registers, control & status registers (program counter, condition codes, status), stack pointer (to the top)), process control (scheduling & state info (process state, priority, scheduling info, event (waiting for)), data structuring (if linked list has pointer to next process), inter process communication, privileges, memory management (pointer to segment & page tables), resource ownership & utilization)
Kernel Functions: process management (creation & termination, scheduling & dispatching, switching, synchronization & inter process communication support, process control blocks), memory management (allocation of address space, swapping, page & segment management), I/O management (buffer management, allocation of channels & devices), support functions (interrupt handling, accounting, monitoring)
Process Switch Steps: save processor context, update process control block of running one, move process control block to appropriate queue, select another process, update selected process control block, update memory management data structures, restore processor context for selected process
OS Execution Types: non-process kernel (user processes on top of kernel), execution with user processes (user processes all have OS functions, all on top of process switching functions), process based OS (OS functions are in separate processes alongside user processes, all above process switching functions)
Thread Parts: execution state, saved thread context when not running, execution stack (user stack plus kernel stack), some static storage for local variables, and access to shared memory and resources of process
Thread Uses: foreground & background work, asynchronous processing, speed of execution, modular program structure
Thread Types: User Level Thread (ULT), Kernel Level Thread (KLT)
ULT Advantages: no need for kernel mode privileges, app specific scheduling possible, OS independent
ULT Disadvantages: can block whole process on system call but thread itself is not blocked (thread blocked status is for waiting on another thread), no multiprocessing
KLT Advantages: opposite of ULT disadvantages, kernel routines can be multithreaded
KLT Disadvantages: thread switch requires switching to kernel mode

Scheduling: important in multitasking & multi user systems, ex: app first then daemon, also deadlines
When to schedule: new process, process exits, I/O wait, blocks on lock, I/O interrupt (resume waiting process or interrupted process?), *generally* when process/thread can no longer continue or activity results in more than 1 ready process
Scheduling Types: long term (add to list of processes to execute), medium term (add to main memory), short term (actual execution), I/O (which request to handle)
Short Term Scheduling Criteria: user oriented (ex: turnaround time, response time, deadline, predictability), system oriented (ex: throughput, processor utilization, fairness, enforcing priorities, balancing resources)
Short Term Scheduling Parts: selection function & decision mode (preemptive does not monopolize and better overall service to processes but more overhead, opposite for non preemptive)
Performance Indices: Arrival, Service (total execution time), Turnaround (total time spent), Normalized Turnaround (turnaround divided by service, 1.0 is best)
FCFS: imple, non preemptive, performs better for long processes, favors CPU bound, performs badly given wildly varied jobs
Round Robin: preemptive, regular interrupt, next process chosen using FCFS, effective for general purpose time sharing systems, short time quantum improves responsiveness, long time quantum improves efficiency (less time switching), still favors CPU bound (fix is to let previously blocked processes finish their time quantum before other ready processes, which is virtual round robin)
Shortest Process Next: non preemptive, choose process with shortest expected running time (need to know), gives minimum average waiting time, possibility of starvation for longer processes, not suitable for time sharing, can be unfair to short processes given varied mix (like FCFS)
Shortest Remaining Time: preemptive, when new process arrives OS will preempt running process if expected time to completion for current is longer than the new one (need to know), long processes may be starved, no bias for long processes like FCFS (is actually against), no additional interrupts like in Round Robin, better turnaround time than Short Process next because short jobs get immediate attention, but higher overhead (need to know elapsed times)
Highest Response Ratio Next: non preenptive, chooses process with highest $RR = \frac{w+s}{s}$, where w is waiting time and s is expected service time (need to know them), after it $RR = \frac{T_r}{T_s}$, considers age of process, if more waiting time response ratio increases, so more likely to be chosen, and longer processes will not be starved
Priority Scheduling: priority per process, can have queue per priority, each queue can have own algorithm, priority boosting needed to prevent starvation (do this for old processes)
Feedback: preemptive, priority 0 is highest, queues per priority, new process start at 0, when preempted or blocked priority reduced, FCFS for each queue (round robin for last), can starve long processes (partial fix: increase time quantum for lower priority queues (ex: priority k can get 2^k time), also possibly do priority boosting after waiting for some time)
Fair Share Scheduling: decide based on process group/user instead of individual thread/process, give fair share to each group, give fewer resources to group with more than fair share & more to group with less than fair share, scheduling done on basis of process priority, recent processor usage of process & group