PCB (Process control block): A structure outside of the process for the OS to keep track of all the information regarding the process such as address space, registers, program counter, etc., Scheduler: The OS / hardware necessary to decide which process to run and when, Fork \rightarrow initiates a new child process from the line after the fork command. Returns pid of child to parent, 0 to child, and -1 if failed

 $\operatorname{Exec} \to \operatorname{runs}$ a different program. Does not return if successful, returns -1 if failed

Wait(int * w_status) \rightarrow called by a parent to wait for some child process to finish. Returns child pid and houses the status of execution in w_status

pipe → initializes a pipe to be used by two different programs to allow information to be passed. Returns -1 on fail.

 $\operatorname{dup} \to \operatorname{changes}$ where the file descriptor points. Returns -1 on fail.

Pipes A pipe allows communication between processes. Reads from p[0] and writes to p[1].

Time-sharing multiple processes run together on a single machine as though they are in sole control

Multiprogramming when process waiting for I/O, OS can have another process use CPU

Multitasking each process gets a time slice, a time limit before being forced off CPU

User mode (CPU) does not have access to read/write I/O devices, or use memory outside its space,

Kernel mode (CPU) Has full access to read/write I/0 devices, and use any memory in the computer

Privileged instruction These are the intructions that user mode can not call

System call A way for user mode to initiate a priviledged instruction

Trap System calls cause a trap which the OS handles, doing the instruction most likely, and then goes back to User Mode

Preemption The notion in scheduling when a context switch will happen without a process finishing

Turnaround time: $T_{completion} - T_{arrival}$, Response time: $T_{first_response} - T_{arrival}$,

FIFO: First In First Out: Simple, no preemption, relies on arrival time, SJF: Shortest Job First: priority queue based off completion time, based on arrival time, STCF: Shortest Time-to-Completion First: priority queue based off time to finish, will preempt if another job shows up with shorter time to completion. Can starve long jobs, RR: Round Robin: Optimal Response time, will simply cycle each process through a time slice of a certain size. Does not give any priority, Lottery: Each process gets a random amount of tickets (can be inveresly proportional to completion time) that the OS then picks a random ticket and gives that process the time slice, Stride: Deterministic solution to lottery, requires states, gives each process a stride amount based off time to finish. grabs the process with the smallest stride and increments the processpass count by the stride of the process., MLFQ: 1) If Priority(A) > Priority(B); A runs, 2) If Priority(A) = Priority(B); A&B run in RR using time slice of given queue, 3) When process enters it starts at highest priority \rightarrow Feedback 4) Once a process has used its time slice its priority gets reduced \rightarrow Confront gaming the system and adding feedback 5) After some time period S, move all processes to topmost queue \rightarrow Priority Boost CFS: Linux Scheduler: uses sched_latency (amount of ms for all processes to run), min_granularity (min amount of time for one process to run), n (number of process), nice score (more nice = less weight) to decide that time slice k = sched_latency * weight k / (weight 0 + weight 1 + \cdots + weight n-1) and vruntime k = vruntime k + runtime * weight 0 / weight k; thus vruntime is counted less per shed_latency with higher weight even if physical runtime is longer. Uses red black tree to self balence, and any new process gets the smallest vruntime of any of the existing processes

Time-slice (quanta in RR) The amount of time for each process to run before a context switch

Oracle (requirement to see into the future) Used by SJF and STCF to decide the future runtime of the process, and CFS

Tickets (Lottery) How the OS decides what process to use

Starvation If a process does not get to run. STCF leads to starvation, FIFO can in rare cases, Lottery can in rare cases

Priority boost (MLFQ) After S time, return all processes to the highest priority

Virtual runtime (CFS) The actual time counted instead of physical runtime

Nice value (CFS) Higher nice is lower weight

Program code A section of memory for the instructions and any static variables

Heap A section of memory for allocated variables (malloc())

Stack A section of memory for initialized variables (int x;)

Memory virtualization The process of transforming messy physical memory into pure virtual memory

Base and Bound Base = the start of the physical address location, bound = max address location, can be virtual or physical. A problem is that if a program needs to use more memory than originally allocated, a lot of updates need to be done.

MMU (Memory Management Unit) The part of hardware responsible for address translation

Segmentation The act of splitting physical memory between multiple sections and/or pages, not making one continguous slot of memory

Segment (address space) Independent parts of the segmentation of physical memory, normally made into a page

Sparse address space A mostly unused address space

Extern fragmentation Gaps in the segmentation that are not used and unable to fit a full segment

Internal fragmentation Unused memory that is allocated by an allocator

Best fit (free-space management policy) A way to decide how to fill segements, will look for the smallest one that still meets the memory requirement.

Paging (memory) A process of creating fixed space units for memory

Page table A table that converts virtual memory to specific pages in memory

Page A fixed size of memory

Frame The physical space for a page of memory

VPN (virtual page number) If total address is 2^m and page size is 2^n VPN is m-n bits, which describes what page number the address is at

PFN (physical frame number) If total address is 2^m and frame size is 2^n PFN is m-n bits, which describes what frame number the address is at

Offset (from start of page) The rest of the bits used to find that exact section of memory

Physical address = virtual address + offset

Valid bit (page table) A bit stating if the memory in the page table is able to be used (may not be all of the memory)

TLB (Translation-Lookaside Buffer) A cache of page table lookups to makes address translation much faster

TLB cache entry A line in the TLB that translates VPN to PFN, along with a valid bit

Valid bit (TLB cache entry) Shows if the cache line is valid, does not say anything about the page table entry valid bit

Memory hierarchy Registers, L1, L2, L3, RAM. The division of memory into specific areas depending on how likely it is needed

Present bit (swap) If the current page is in memory (1) or on disk (0)

Page fault When the page asked for is not present

High watermark The amount of free memory to stop swapping from low watermark

Low watermark The minimum amount of free memory, if under will start pushing pages to swap space

Reference string A sequence of attempted accesses to model their accesses in different replacement policies

OPT (optimal) The optimal cache removal based off the reference string

FIFO (first in first out) As it sounds, the first one in is the first one out on a miss with no free space

Random Randomly pick a entry for each miss with no free space

LRU (least-recently used) Remove the least recently used when a miss with no free space

Clock Algorithm An attempted LRU without lookups to see which was last used. Add a "use bit" to each entry of the cache. Set 1 when line is used. round robin, setting to 0 when 1 is found, otherwise removing the first 0

Belady's Anomaly How in FIFO there are cases when larger cache means more misses

Dirty bit (swap) A bit that starts at 0 and is set to 1 on a write, to know if the page needs to go to swap or not

Thrashing When there is more time spent on page-faults than execution

Threads Multiple points of execution of a program within a single process, shares data and a heap with other processes

TCB (Thread Control Block) A structure for the OS to keep track of different threads

Concurrency vs Parallelism Concurrency is running one after another on one CPU, parallelism is running items at the same time on different cores

Race condition When two or more threads step on each other leading to undesireable or weird effects

POSIX Portable Operating System Inferface, standards for OS APIs

pthreads a specific api for thread creation and synchronization

pthread_create(pthread_t *thread, const pthread_attr_t attr, void *(*start_routine) (void*), void* arg) Creates a new thread with thread pointing to a buffer to store the id of the new thread, attr describes the attributes of the new thread, starts by invoking start_routine arg is a pointer to arguments for start_routine. Returns 0 on success, error number on error

pthread_exit() Terminates calling thread

pthread_join(pthread_t th, void **retval) Wait for a thread to terminate and will add the exit status to the location at retval if not null. 0 on success, error number on error. also deallocates TCB of whatever thread terminated

pthread_mutex_lock(pthread_mutex_t * mutex, const pthread_mutexattr_t * mutexattr) initalizes a mutex lock, mutex points to the lock, second attribute describes the lock, returns 0 on success, error number otherwise

 $pthread_mutex_t *mutex) \ destroys \ the \ mutex, \ attempting \ to \ destroy \ locked \ mutex \ results \ in \ undefined \ behavior \ Mutual \ exclusion \ Only \ one \ thread \ at \ time$

test-and-set A process of determing if a lock is locked or unlocked by testing the value of a pointer, and setting it to a new value Spin lock A system of waiting for a lock to unlock with a while(locked);

Spinning loops What is imployed in a spin lock

pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m) waits for signal to condition c, releases m in while waiting pthread_cond_signal(pthread_cond_t *c) Signals on the condition, waking waiting threads

Producer/consumer (bounded buffer) problem Consider 2 conumers and 1 producer. Producer puts one 1, then signals that item has been placed. If solved incorrectly both c1 and c2 will attempt to get, causing an error. p1 could also attempt to put on a full buffer. Fix is to signal both empty and full.

Semaphore Provides both locking and signaling

sem_init(sem_t *s, attr, num of resources) must init with number of resources, generates a new semaphore at s with attributes and number of resources as described.

sem_wait(sem_t *s) will wait until s has at least 1 resource

sem_post(sem_t *s) will free up one resource of s

Binary semaphore Using semaphore as locks, having only two states, locked and unlocked. thus binary.

Reader-Writers problem Multiple people can read from data at one time, but only one writer can write with no readers allowed.