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., pipe() → initializes a pipe to be used by two different programs to allow information to be passed. Returns -1 on fail., dup() → changes where the file descriptor points. Returns -1 on fail. Trap → System calls cause a trap which the OS handles, doing the instruction most likely, and then goes back to User Mode.
Turnaround time: $T_{completion} - T_{arrival}$, Response time: $T_{first\_response} - T_{arrival}$,
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, Lottery: Each process gets a random amount of tickets (can be inverse proportional completion time) that OS picks a random ticket to give 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 processṕass count by the stride of the process., 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
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
Base and Bound Base = the start of the physical address location, bound = max address location, can be virtual or phyiscal. A problem is that if a program needs to use more memory than originally allocated, a lot of updates need to be done.
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.
Physical address = virtual address + offset
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, Present bit (swap) If the current page is in memory (1) or on disk (0)
High watermark The amount of free memory to stop swapping from low watermark, which is the minimum amount of free memory
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
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
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, errror number on error, 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 succcess, error number otherwise, pthread_mutex_unlock(pthread_mutex_t *mutex) destroys the mutex, attempting to destroy locked mutex results in undefined behavior, 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, Spinning loops What is imployed in a spin lock, which is looping for a 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(p
*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.
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,