2) **ProcessManagmnt**: kernel vs user: kernel: privileged instructions (set mode bit), direct access to all of mem, to dvices. Sys calls: only interface from program 2 os. Traps: generated by cpu as result of error, like involuntary sys call. Interrupts: generated by device that needs attention. Process: program in execution, execution stream in context of process state. Process state: registers, address space, heap stack, open files. Address space: static: code & some global vars; dynamic: stack & heap. Why dynamic alloc: dk memory needed at compile time, if static must be pessimistic, recursive procedures, complex data structures. Stack: efficient, simple: pointer separates allocated & freed space, increment pointer 2 alloc, decrement 2 free, no fragmentation, management done automatically. Heap: allocate from random location: has allocated areas & holes, order of alloc & free unpredictable; work for all ds, alloc slow, fragmentation. PID=fork(): creates identical copy of parent, in parent returns pid of child; in child returns 0. Exec(filename): loads executable from file w filename. Wait(): wait for 1 of children to terminate. Exit(): terminate process. Shell operation: new command line(!=logout): shell forks a new process & waits, child execs prog on command line. Linux processes tree: Boot: 1st process after boot is init; user logs in: init forks & wait, child execs shell; user runs make: shell forks & waits, child execs make; another user: init forks & wait, child execs shell; make runs gcc: make forks & wait, child execs gcc; gcc finishes: gcc exits, make returns from wait; 2nd user logs out: csh exits, init returns from wait; make runs cp: make forks & wait, child execs cp. Process (for os): computes (use cpu) or does IO (use devices); Issues (single): low utilization, long wait times. Multi: state diagram: new->ready->running-> terminated/waiting[IO]->ready[IO completion]. Process switch: switch from 1 process running on cpu to another, st can later switch back to process currently holding cpu. Process consists of code stack heap (in process private locations) & registers, mmu info (in shared locations). Ex: P1->P2: save registers-> PCB[P1].saveArea, restore PCB[P2].saveArea->registers.