2/17/2023 : Processes

Process
- running program
- multi core processors can run many processes @ the same time

Program "state"
- CPU registers
- Program counter
- Stack counter
- Right now
- can leave program stopped for arbitrary time

- 1 process per CPU (generally)
  - hyper-threading does weird stuff

Address space
- space in memory were program lives
- "Core Wars"
- ideally memory is protected
- makes program think its alone
  - loader, part of OS that gives space to program
  - processor can give virtual memory
  - special registers that indicate address space

Memory Heiarchy

Cache: faster than DRAM, not as fast as a register
- L1: small, on CPU, fast, expensive
- L2: may be on or off chip
- L3: offchip, SRAM

main memory
- DRAM, medium speed
- Disk: slow

Memory manager handles memory

Temporal & spacial locality
- LRU principle
- "best predictor of near future is the recent past"

Old days
Fixed partitions
- separate input queues for each partition
- Big computer, run processes in dedicated memory partitions

Multi Programmed system performance (multiprocessing)
ex) Three processes on 1 cpu, 3 times as long to do 1 process
- why does it work?
- your computer does a lot of waiting, waiting for I/O
- context switch time
Need:
- memory relocation (adjust locations of address when running it in a diff location
- memory protection (keep memory from being overwritten)

Base & limit Registers
- Base where program starts
- limit, where program ends
- throws seg faults
allocating memory
- remember first choice, next choice, "best" fit, Worst fit
- next choice is best
- external fragmentation, small slivers of data, a problem w/ "best" fit

Freeing memory:
- easy w/ bitmaps
- linked lists: modify adjacent elements as needed
- merge regions as needed

Memory Fragmentation

- slivers of unused memory that were unmerged

Buddy allocation
- chop up memory in orders of 2
- when chunk is freed, see if it can be merged w/ buddy
- fast

Slab allocator
- similar to Buddy allocation

Virtual memory
- divide program into pages of same size
- don't need a limit + base register for each page bc you know how big the page is
- gives out more memory than exists,
- How? Your <u>entire</u> program isn't running all @ once, can load a small part of program memory.
- slower
- hide from process, thinks it starts from 0
- Operating system handles everything
- parts of programs put to sleep while retrieving memory

Programs have virtual memory, memory management (MMU) handles translation to physical address

virtual page mapped to physical memory

Page table
- each page has:
  - protection
  - dirty bit: has page been modified
  - ref bit:
  - valid Bit: does it exist in physical memory
  - page frame #

every process has it's own page table

can share pages by pointing at same location

Unix: fork

makes clone of program

Processes can end voluntarily or involuntarily

Process group
- zombie process
- group lead process

Process states
- created
- waiting