Process

- code, data, stack (own address space usually)
- program state (CPU registers, program counter, stack ptr)
- only 1 process runs at a time
  - multi-core CPU multiple processes
- loader relocates instructions by ^address (base) to each other
  - ↳ processor & OS provide virtual memory

Memory Hierarchy

- cache ⇒ L1 (CPU chip), L2 (on/off chip), L3 (off chip SRAM)
- main memory (DRAM)
- disk

- memory management goal: lay components out in memory

Fixed Partition: Multiple Programs

- fixed partitions
  - divide memory into fixed space
  - assign process to space when free
- mechanisms
  - separate input queues for each partition

Multiprogramming

- needs relocation & protection
- OS can't be certain where a program will be load in memory
  - ↳ must keep memory separate

Base & Limit Registers

- base: start of process's memory partition } contained in registers
- limit: length of process's memory partition }
- physical address = location in actual memory
- ~~logical~~ address = base + logical address
- logical address = location from process's POV

(error if logical > physical)

- Allocating Memory; searches thru region list to find large space
  - first fit = first suitable hole on list
  - next fit = first suitable after previously allocated hole
  - best fit = smallest hole larger than desired region (wastes least space)
  - worst fit = leaves largest space

- Freeing memory: - easy w/ bitmaps
  - modify adjacent elements in linked list
    - merge adjacent free regions to 1 region

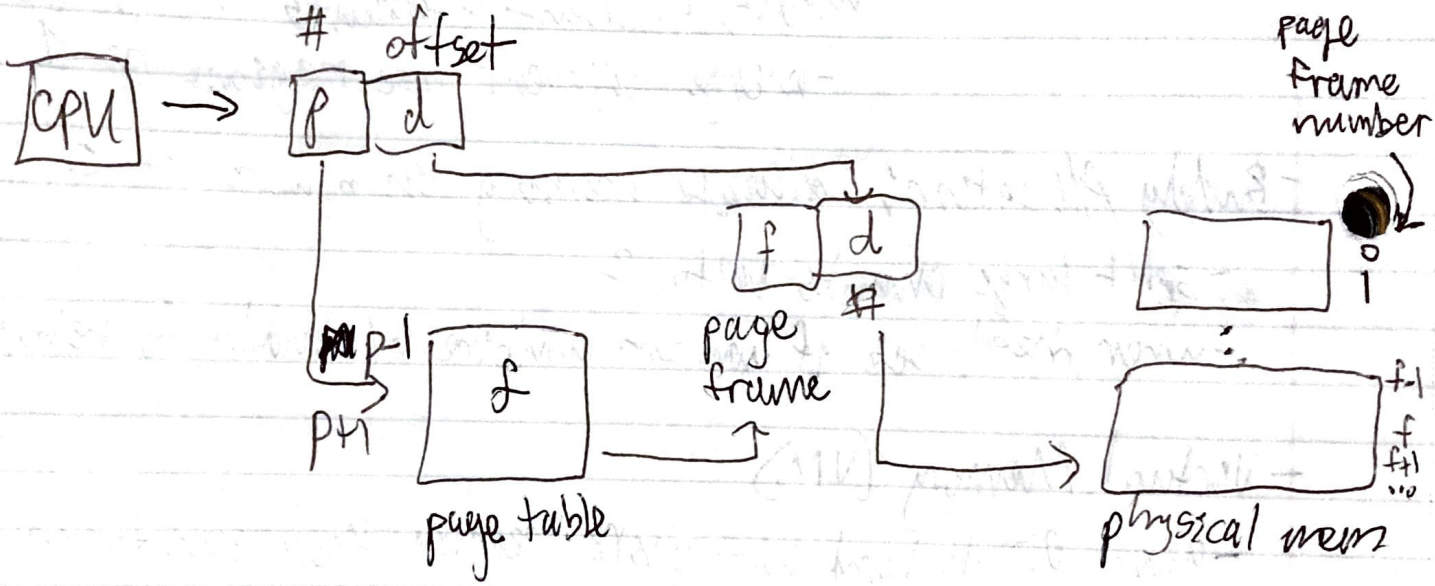
- Buddy Allocation; allocate memory in powers of 2
  - split large chunks into 2
  - when freed, see if you can combine w/ buddy to rebuild

- Virtual Memory (VM)
  - allow OS to hand out more memory than exists on system
  - keep used in physical memory, move rest to disk

- virtual address mapped to physical addresses = paging
  ↳ page table translates virtual page number to physical page number (but not all VM has physical page)

- Page Entry: - valid bit (has corresponding physical frame)
  - page frame # (page in physical memory)
  - referenced bit (if data was accessed)
  - dirty/modified bit (if modified)

Mapping Process: - split address into page # p & offset d
(page size = 2^d bytes)
- page #: - index into table
- contains base address of page in phys mem
- page offset: added to base address to get actual physical mem address



Process Model
- conceptual model of multiprogramming 4 programs (independent, runs sequentially)
- only 1 process active at any instant
- can be short, only applies if single CPU in system

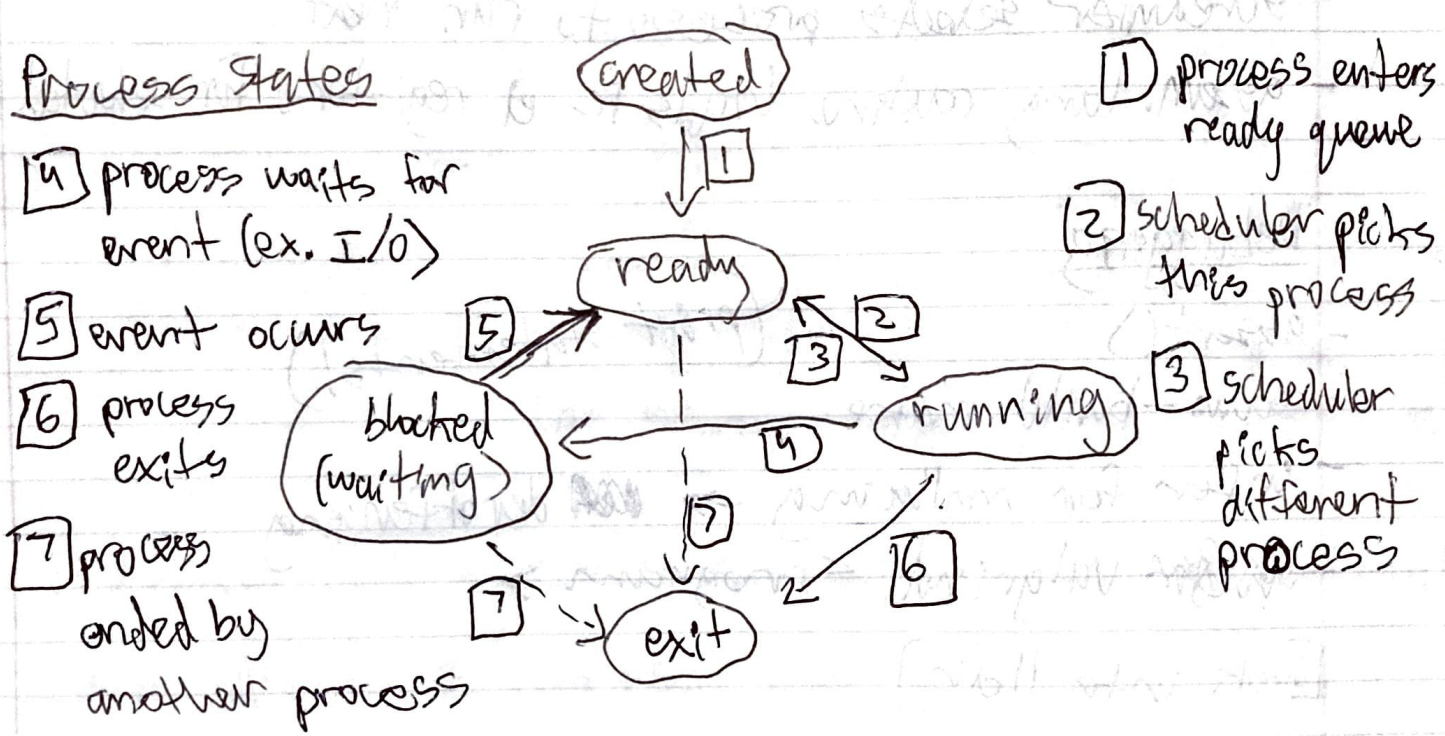
- process created by ① system initialization
② execution of process creation system call

- Process ends: ① voluntary (normal or error exit)
② involuntary (fatal error / killed by another process)

Process Hierarchies

- parent creates child process, child can create own children
- Unix forms hierarchy "process group"
  - if terminated, children inherited by terminated process's parent

Process States



Process Table Entry

registers
program counter
CPU status word
} may be stored on stack

Trap/Interrupt

- hardware saves program counter (stak/register)
- hardware loads new PC & identifies interrupt
- assembly lang routine saves registers
- " " sets up stack
- " " calls C to run service routine
- service routine calls scheduler
- scheduler selects process to run next
- assem. lang routine loads PC & registers for selected process

Debugging

- assert() (print statements!)
- scan-build make
- infer for malloc → dereffencing
- valgrind <program>

[look into lldb]

Hashing & Bloom Filters

- binary search in sorted array = $O(\log(n))$
- hashing
  - $O(1)$ search time on average, $O(n)$ worst case

- Hash Table (HT)
  - unordered collection of key-value pairs
  - efficient lookup, insert, & delete operations

⇒ hash(key) = index ∈ [0, m-1] w/ m slots

- function uses all input data
- computation is fast
- minimizes hash collisions

- division based: $h(k) = k \bmod m$ (key) (table size)

- string-valued: take characters from key, compute int using method, int mod table size

- load factor (λ) = # of slots filled out of total slots in HT

- hash collision = 2 data pieces have same hash value
  ↳ rare in good hash funcs w/ proper range
  but unavoidable if # of keys > # of hash values

[linear probing, quadratic probing, double hashing]

- Cache performance
  - chaining = bad as keys are stored in linked lists
  - open addressing = good as everything stored in same table
- Space
  - chaining uses extra space for links
    - part of HT never used
  - no links in open addressing, slot can be used even if input isn't mapped to it

- Bloom Filter (BF)
  - data struct, tells if element in set ⇒ bit array
  - ex) $k = 3$ hash func
    pass elements $\{x, y, z\}$ thru all 3
    and set corresponding bits in BF
  - check if key exists in HT
  - returns possibly in set or definitely not in set
    ↳ false (+) probability = $\left(1 - e^{\frac{-kn}{m}}\right)^k$
    (n = # of inserted elements, m = # of bits in array, k = # of hash functions)
  - $O(k)$ hash func, space of what holds data $O(m)$