Midterm 1 Questions

Understand process scheduling and understand how long time it would take to execute jobs sequentially vs. in parallel?

- Metrics
  - Turnaround Time: Time Finish - Time arrival
  - Waiting Time: Time spent in queue
  - Response Time: Time Start -Time Arrival
- Algorithms
  - FIFO - First in First Out
  - SJF - Shortest Job First
  - STCF (Preemptive Shortest Job First)
  - Round Robin
  - Proportional Share Ticketing System
    - Give each process a certain amount of tickets based on how important it is
- Multi-Processing
  - Single Queue Multiprocessing
    - All CPUs pull from a single global queue
    - Can be implemented using cache affinity, having one processor execute more of a certain process
  - Multi Queue Multiprocessing
    - Places each job in exactly one scheduling queue
    - Each queue follows its own particular scheduling discipline
    - Provides scalability and cache affinity
    - Migration: Moving a process from one CPU to the other
- Question from the book: https://cs.stackexchange.com/questions/51432/operating-system-processes
- Time to complete process = CPU time required/CPU Utilization
- CPU Utilization = 1-(IO Wait)^number of processes

Understand virtual memory and virtual to physical mapping.

- Benefits
  - Easy to use
  - Memory efficient
  - *Guaranteed isolation of processes and OS*
- Virtual Memory Designs
  - Base and Bound approach: Uses a base(start) and bound(end) address for contiguous block of virtual memory (code, heap, and stack), these must be saved when a context switch occurs
    - Down side: big chunks of virtual memory may not fit into physical memory
  - Segmentation approach: Virtual memory is split into 3 portions (instead of being contiguous). Each portion (code, heap, and stack) has its own base and bound

address. All three portions are still associated as the same block of virtual memory, but are spread throughout physical mem

- Virtual Memory Mapping
    - Virtual memory addressing starts at 0 and goes to the size of the memory
        - Ex: a block of virtual mem of size 4k would have a virtual address of 0-4096
    - Virtual memory can be mapped anywhere in physical memory
        - Ex: physical base address of 40k == virtual base address of 0
    - Virtual address offset is not based on physical memory, it is added to the base of the virtual memory
        - Ex: virtual offset of 100 will be at vm base + offset: (0 + 100)
    - Calculating the physical address of virtual memory segment.
        - 1. Get physical memory of virtual address base: 42k = 0
        - 2. Add offset in virtual memory to base of physical memory: Offset = 400 so physical address id 42k+400.
        - Can be rearranged to calculate the offset or the virtual memory address
    - Segment can be identified by the top few bits of the memory address
    - Extra hardware support is need to:
        - Keep track of which way the stack and heap grow
        - Keep track of permissions for segments (read, write, etc) to allow for code sharing
    - Types of segmentation:
        - Fine grained: more segments of smaller size, requires segment table
        - Coarse grained: small number of segments of larger size
    - Fragmentation: small portions of memory between segments that are too small to ever be allocated
        - Solution: Compaction- copying the memory on either side of the small unallocated memory elsewhere, and then recopying it back to a location but with the allocated memory butting up against each other, allowing no space in between. Repeating this will cause all the allocated memory to be block together, same with unallocated

Understand how the OS maintain free disk block list in memory

- A free list contains a set of elements that describe the free space still remaining in the heap.
- Splitting: locating a free chunk of memory that can satisfy the request and splitting it into two. The first chunk it will return to the caller; the second chunk will remain on the list.
    - Splitting is used when the request is smaller than the size of any particular free chunk.
- Coalescing free space when a chunk of memory is free.
    - When returning a free chunk in memory, look carefully at the addresses of the chunk you are returning as well as the nearby chunks of free space; if the newly

freed space sits right next to one (or two, as in this example) existing free chunks, merge them into a single larger free chunk.

- Headers are assigned to newly allocated regions, to store the size of a region and a magic number.
    - When a user requests N bytes of memory, the library does not search for a free chunk of size N; rather, it searches for a free chunk of size N plus the size of the header
- Strategies for managing free space:
    - Best Fit
        - Tries to reduce wasted space, but pays a heavy performance penalty when performing an exhaustive search for the correct free block
    - Worst Fit
        - leading to excess fragmentation while still having high overheads
    - First Fit
        - advantage is speed – no exhaustive search of all the free spaces are necessary – but sometimes pollutes the beginning of the free list with small objects
    - Next Fit
        - The idea is to spread the searches for free space throughout the list more uniformly, thus avoiding splintering of the beginning of the list. The performance of such an approach is quite similar to first fit, as an exhaustive search is once again avoided
    - Segregated lists
        - The basic idea is simple: if a particular application has one (or a few) popular-sized requests that it makes, keep a separate list just to manage objects of that size; all other requests are forwarded to a more general memory allocator
        - fragmentation is much less of a concern; moreover, allocation and free requests can be served quite quickly when they are of the right size, as no complicated search of a list is required
    - Buddy allocator
        - this scheme can suffer from internal fragmentation, as you are only allowed to give out power-of-two-sized blocks
        - Advantage is in coalescing the free chunks

Explain how an OS can facilitate installation of new device without any need for re-compiling OS

- http://www.csd.uoc.gr/~hy345/assignments/2013/cs345_recitation3_sol.pdf
- (Unix Specific)
- The OS will facilitate the installation without recompiling the OS
- There is a table indexed by device number

- Each table entry contains pointers to the following functions: opening, closing, reading, writing, and other functions
- A new entry is made in this table and the pointers filled in, to load the new driver

Given disk rotation speed, number of sector/cylinder, etc. How long does it take to read a sector.

- Problem 1: https://cs.nyu.edu/~mwalfish/classes/ut/s12-cs372h/hw/sol6.html
- https://www.cs.colostate.edu/~cs430dl/yr2015su/more_examples/Ch9/Exercise9_5.pdf
- Chapter form book: http://pages.cs.wisc.edu/~remzi/OSFEP/file-disks.pdf
-