Q1

(a) [3 Marks] Explain the difference between compile time, load time and execution time binding, specifically talk about how the assignment of memory addresses changes limitations on program loading.

Compile time binding means that specific memory addresses are assigned to the program as it is compiled into an executable, this requires knowing how much memory the process will need when it reaches runtime and how that memory will be used. Code will need to be recompiled if the starting location changes.

Load time binding means that absolute memory addresses are assigned as the process is moved to RAM for execution. The compiler will generate relocatable addresses which the loader will translate to absolute addresses.

Both compile time and load time binding require static linking and loading.

Execution time binding means that absolute memory addresses are not assigned until the program begins execution, thus the process can be moved around memory addresses during execution and allows dynamic loading/linking.

(b) [2 Marks] For a page table with 1K entries, work out how much extra memory we need to set aside (overhead) for set-ordering and second chance algorithms.

Set-ordering will need 2 bytes per entry so 2KB will be needed.
second chance algorithms only use 1 bit per entry so 1000 bits

(c) [4 Marks] Explain how sharing is accomplished in a paging system.

In paging systems processes can share read-only pages but not data pages. When a process loads if its code is already loaded into physical memory then the page table can map to the same locations in physical memory.

(d) [4 Marks] Explain how sharing is accomplished in a segmented system.

Sharing segments is similar to sharing pages except the segment table can map to entire segments rather than the individual memory pages.

(e) [4 marks] Describe when a page replacement algorithm can be called a stack algorithm. Explain why the least-recently-used (LRU) page replacement algorithm is a stack algorithm.

A page replacement algorithm is a stack algorithm when the pages in memory for N frames is always a subset of the set of pages that N + 1 frames would hold. LRU moves pages to the top of the stack when they are referenced thus the top n pages are always the *n* most recently used pages, if the number of frames is n+1, the top of the stack will be the n+1 most recently used pages.

Q2

[4 marks] What is a copy-on-write feature in virtual address systems and under what circumstances is it beneficial to use this feature?

Copy-on-write means that processes will read from shared pages but when attempting to write to a page processes will be provided a personal copy. This is useful for situations such as a fork, where the process's full address space might otherwise need to be copied.

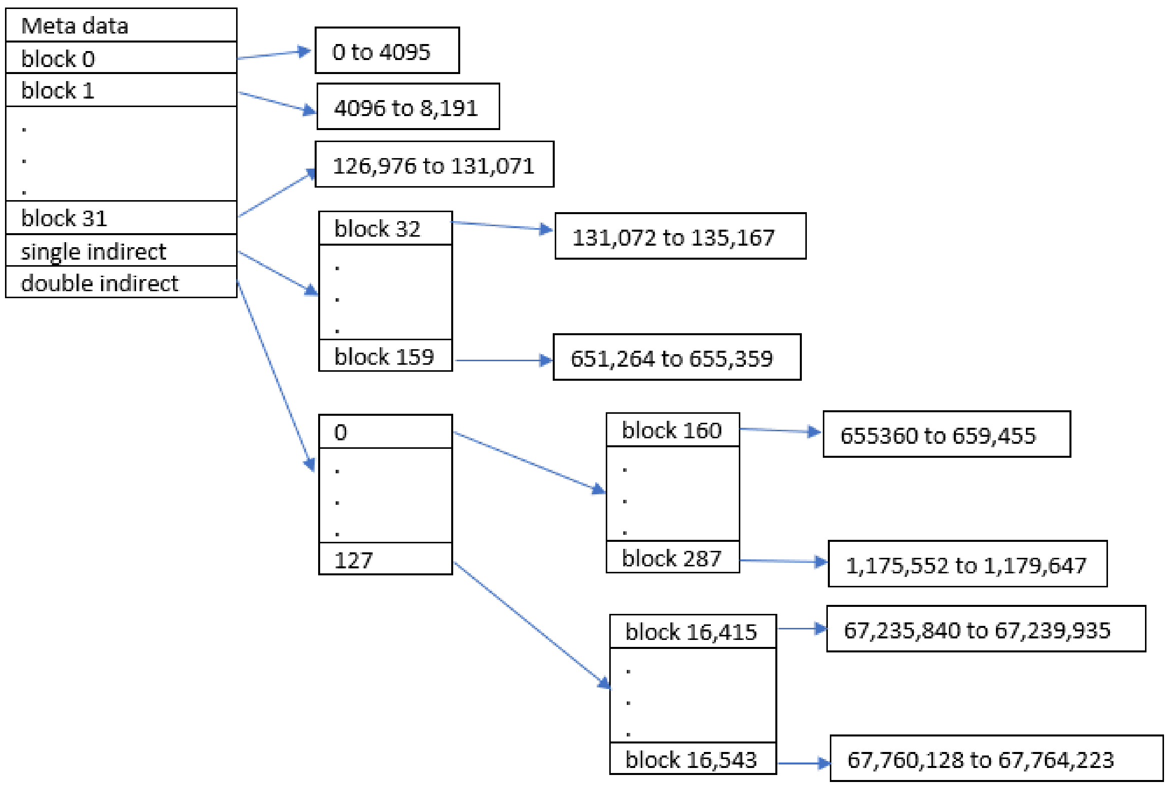
Q3

Assume we have a file system implemented with 32 bit file addresses and 4Kb blocks. It uses inodes with 32 direct blocks, 1 single indirect and 1 double indirect. Assume that any files are already open, but no references have yet been performed.

(a) [4 Marks] Draw the inode structure and indicate the addressing ranges for each of the block pointers and how many bytes of the inode are used for the block pointers.

4Kb = 4096 bytes
4096 / 32 = 128 pointers per block

direct blocks + single indirect + double indirect
(32 * 32 bits) + 32 bits + 32 bits = 1088 bits
= 136 bytes of the inode are used for the block pointers



(b) [2 Marks] How many disk accesses are required to read byte 101,248? Be sure to explain your reasoning.

101,248 / 4096 = 24.7
therefore byte 101,248 is in block 24 this is in the direct blocks so the number of disk accesses is 1 + 25 = 26

(c) [2 Marks] How many disk accesses are required to read byte 4,231,901? Be sure to explain your reasoning.

4,231,901 / 4096 = 1033.2
therefore byte 4,231,901 is in block 1033 this is in the double indirect block
(1033 - 159) / 128 = 6.9
therefore byte 4,231,901 is in the 6th single indirect block of the double indirect block
1 + 7 + 1034 = 1042

(d) Given the following queue of disk accesses [41, 92, 54, 26, 3], a disk of 100 cylinders, a linear seek time of 0.8ms/cylinder, a starting position of 13, and a scan direction from 0 →100:

(i) [3 Marks] Calculate the total time spent seeking for a SSTF schedule.

Seek path: 13, 3, 26, 41, 54, 92
Total seek time: ((13 - 3) + (26 - 3) + (41 - 26) + (54 - 41) + (92 - 54)) * 0.8 = 79.2ms

(ii) [3 Marks] Calculate the total time spent seeking for a SCAN schedule.

Seek path: 13, 26, 41, 54, 92, 3
Total seek time: (((26 - 13) + (41 - 26) + (54 - 41) + (92 - 54) + (92 - 3)) * 0.8 = 134.4