OS Assignment #7: Review Questions

Leiqing Cai	lc9ac@virginia.edu
Jared Culp	jjc4fb@virginia.edu
Alex Hutcheson	amh4bm@virginia.edu
Bryan Walsh	bpw7xx@virginia.edu

1. *Consider a logical address space of 64 pages of 1024 words each, mapped onto a physical memory of 32 frames.*
    a. *How many bits are there in the logical address?*
    16 bits (6 for page number, 10 for page offset)
    b. *How many bits are there in the physical address?*
    15 bits

2. *Describe a mechanism by which one segment could belong to the address space of two different processes.*

For a segment to be in the address space of two different processes, each process would have to have an entry in it's segment table with the same base pointer. In that case, both segment table entries would point to the same physical location in memory.

3. *Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs.*

    A page fault occurs when a process tries to use a valid page that is not currently mapped in physical memory. The operating system deals with this as a trap and responds in the following way:

    - The OS saves the state of the interrupted process when the page fault occurred
    - The OS resets hardware pipelines and transfers control to the page fault handler
    - The OS checks the page table for this process and looks to see if its reference was a valid or invalid address. If it was invalid, then the process gets terminated by the OS. If it was valid, then the OS prepares to read the valid page into a physical address
    - The OS schedules a disk operation to read the page into physical memory. The process enters a blocked state.
    - The OS reads the page into memory and then updates the page table to show that the page is now in memory
    - The OS changes the process' state to ready

- The OS resumes the process at the line at which the page fault occurred and the process resumes as though the page fault never occurred

4. *We have an operating system for a machine that uses base and limit registers, but we have modified the machine to provide a page table. Can the page tables be set up to simulate base and limit registers? How can they be, or why can they not be?*

   The page table can be used to simulate base and limit registers, but we will have to constrain ourselves to using only fixed-size segments (with the size fixed to the machine's page size). The base pointer is placed in the segment table, and the limit register is implicitly assumed to be the page size.

5. *Explain the usefulness of a modify bit.*

   The modify bit tells us whether or not a page has been modified. If it has been modified then we must save the changes of the page being swapped out to the disk. If it has not been changed then we can immediately overwrite memory with a new page without writing the old page to disk.

6. *Explain why SSTF scheduling tends to favor middle cylinders over the innermost and outermost cylinders.*

   The SSTF algorithm always chooses the closest available cylinder, and the center of the disk has the smallest average distance to all other cylinders. The result is that the algorithm tends to move towards the middle, rather than towards the extremes.

7. *Could you simulate a multilevel directory structure with a single-level directory structure in which arbitrarily long names can be used? If your answer is yes, explain how you can do so, and contrast this scheme with the multilevel directory scheme. If your answer is no, explain what prevents your simulations success. How would your answer change if file names were limited to seven characters?*

Yes, you could simulate a multilevel directory structure within a single-level directory. In this scheme, a "directory" would simply be a special type of file. This file would contain a list of all of the files contained in that directory, some of which may be other directory files. A program traversing a directory tree would open up that directory's file and read the list. In order for there to be no collision between files in different directories that have the same name, each filename would be the full path to the file (ex. "/home/amh4bm/docs/setup.conf"). All files (both "normal" files, and the special "directory" files) would exist within the single-level directory on which the system is implemented.

Due to this full-path naming convention, a seven character limit would be a severe restriction on this scheme. Although it would still be technically possible, the directory structure would be limited to a maximum depth of 6 directories (since the 7th character would need to be used for filenames in the directory). If a directory-separator character must be used in the file path, then

the maximum directory depth is only 3. In either case, files at the lowest level could be named using only a single character, and the maximum number of files in the lowest-level directories would be limited by the number of characters in the character set. All in all, this system does not seem appropriate for an environment with the 7 character limit.
This solution is different from the typical multi-level directory structure. In a normal multilevel directory structure, directories exist as associative arrays relating symbolic file names to their inode numbers. Using the inode number, the kernel is able to directly manipulate the file.
Our simulated solution adds another layer of abstraction to this process. To access a file in a directory, user code reads the list of files from the directory file, which is simply a textual list of files. It then references the “master directory” (the one actually implemented in the underlying filesystem) to retrieve the inode number for that filename and operate on the file.

8. *Why is it advantageous for the user for an operating system to dynamically allocate its internal tables? What are the penalties to the operating system for doing so?*

Dynamically allocated tables allow the tables to grow as needed as long as there is available space. This prevents caps on table size due to statically defined buffer sizes.
However, this comes at the cost of both additional system complexity and resource usage. Dynamically allocated memory is more difficult to reason about and debug, and managing dynamically allocated memory can consume a non-trivial amount of resources.

9. *How does DMA increase system concurrency? How does it complicate hardware design?*

   Direct memory access avoid burdening the CPU with programmed I/O by offloading some of the work to the direct-memory-access controller. As a consequence, the CPU can continue to perform useful processing concurrently while data is being transferred from I/O devices to an preallocated physical memory space.

   Each DMA session takes the following 6 steps:
   1. Device driver is instructed to transfer data to buffer at some address
   2. Device driver tells disk controller to transfer some number of bytes from the disk to that buffer
   3. Disk controller receives request and initiates the memory transfer
   4. Disk controller sends each byte to the DMA controller
   5. DMA controller transfers bytes to buffer, increasing memory addresses and decreasing the number of bytes to transfer until completed
   6. When done, the DMA interrupts the CPU to signal that all memory has been transferred.

   Therefore, the obvious tradeoff here is that more circuits are needed (for the DMA controller and related interface/connection between the DMA controller and main memory and between the DMA controller and I/O devices) in order to accomplish the above stated operations.

10. *What are the main differences between capability lists and access lists?*

Access list and capability lists are two different ways of implementing access matrix. An access list is a list of <domain, right set> pairs for an object. A capability list is a list of objects together with the operations that are allowed to be performed them. Therefore, the main difference is that access lists are object specific, while capability lists are domain specific.

Also, access lists incurs high overhead and capability lists are not inefficient for revocation.