Homework 3 Answers

Due: May 23, 2022 **Points:** 100

When you do the homework, please put the answers to questions 1-4 on one page, and the answers to each of the others on separate pages. You can save this file and put your answers on it. This will make. using Gradescope to grade the assignment much easier than if you submitted everything without regard to pages.

Remember, you must *justify all your answers*.

Short-answer

1. (*6 points*) What is "device independence"?

   Answer: Device independence refers to the procedure of hiding the characteristics of I/O devices from higher level software. For example, the device independent software in a system may make a disk appear as a stream of bytes rather than a device storing blocks of bytes.

2. (*7 points*) What is thrashing?

   Answer: Thrashing occurs when a process or system spends its time servicing page faults and not executing the process.

3. (*10 points*) What is the difference between paging and segmentation?

   Answer: With paging, both the virtual and physical memories are divided into equal sized sets of words. With segmentation, virtual memory is divided into logical parts such as per function and per data structure, and these segments are loaded into memory as needed.

Long Answer Questions

4. (*25 points*) You are the president of Cheapo Computronics, Inc., and your star hardware designer has suggested a brilliant idea: Implement segmentation, but let the least significant m bits of a virtual address be used to select the segment, and let the other bits determine the offset. What is the problem with this idea?

   Answer: Suppose that a program accessed a set of n instructions sequentially, and $n \leq m$. If the designer's idea were used, each access would be to a different segment, which means that a lot more segments would have to be in memory for the program to run than if the usual mode of addressing were used (n mod m rather than $\frac{n}{m}+1$.) This would either increase the amount of time the CPU must spend in moving segments in and out of memory, or decrease the number of processes which could be resident, either of which would lower the degree of multiprogramming unnecessarily.

   Also, since accessing each instruction will require a segment table access, consider the effect of a small associative memory used to speed these accesses. If the associative memory is too small to contain all segment table entries, memory accesses will cost twice as much because two memory accesses will be needed to read (or change) a memory location: one for the segment table entry, and one for the datum. On the other hand, associative memory is expensive too, so increasing the amount of associative memory will increase the price of the computer, besmirching the name of Cheapo Computronics, Inc. So either way, the designer's proposal would adversely impact the machine, and (probably) sales.

5. (*25 points*) This question asks you to compare different disk scheduling policies.

   (a) Under very light loads, all the disk scheduling policies we have discussed degenerate into which policy? Why?

   (b) Consider a system on which a seek takes $0.5 + 0.4T$ ms, where T is the number of cylinders moved. Then assume the arm is initially at cylinder 100, the disk has 200 cylinders, and the arm is moving inward. Will requests scheduled by a FCFS disk scheduling policy ever have a lower mean waiting time than those scheduled by a SCAN policy? Than those scheduled by a SSTF policy? Justify your answers.

Answer:

(a) Under light loads, only one disk request will be in the queues at any given time so all of the scheduling policies degenerate to FCFS. Note however, that not all FCFS policies are equal as SCAN and C-SCAN will both continue to move the head across the entire disk.

(b) Obviously, from part a, there are conditions where the mean can be the same. FCFS can be better than SCAN if the requests arrive in such a way that the algorithm has just passed a track when a request arrives. As an example, consider these requests: 98, 99, 101. FCFS will handle them in order. The mean waiting time is:

$$([0.5+0.4(2)]+[0.5+0.4(3)]+[0.5+0.4(5)])/3 = (1.3+1.7+2.5)/3 = 1.83\text{ms}$$

If the arm is moving inward, SCAN will handle the 101 request, then scan to the edge of the disk and back to 99, so the total seek distance is 202. The mean waiting time is:

$$([0.5+0.4(1)]+[0.5+0.4(201)]+[0.5+0.4(202)])/3 = (0.9+80.9+81.3)/3 = 54.37\text{ms}$$

Thus, FCFS can be better than SCAN.
A similar example for SSTF is possible. In this case, the sectors have entered the queue in this order: 103, 98, 90. FCFS will handle these in order. The mean waiting time is:

$$([0.5+0.4(3)]+[0.5+0.4(5)]+[0.5+0.4(13)])/3 = (1.7+2.5+5.7)/3 = 3.30\text{ms}$$

SSTF will handle the requests in the following order: 98, 103, 90. The mean waiting time is:

$$([0.5+0.4(2)]+[0.5+0.4(5)]+[0.5+0.4(18)])/3 = (1.3+2.5+7.7)/3 = 3.83\text{ms}$$

So given an appropriate queue of requests, FCFS can be better than any specified scheduling policy except an optimal scheduling policy.

6. (*27 points*) Consider a file currently consisting of 100 blocks of 512 words each. Ignoring the access to update the device directory, and assuming a disk block number fits into a single word, how many disk I/O operations are involved with contiguous, linked, and indexed allocation strategies, if one block:

   (a) is removed from the beginning?
   (b) is added in the middle?
   (c) is added at the end?

   Answer:

   (a) With contiguous allocation, only the position of the file on the device must be updated, so there are no disk I/O operations involved. With linked allocation, the first block needs to be read to obtain the address of the second block, and then the pointer to the file must be updated; the read is 1 disk I/O operation. With indexed allocation, the index block must be read, updated, and written out, so two I/O operations are needed.
   (b) With contiguous allocation, the blocks trailing the new block must be shifted; in the best case, this requires 101 disk I/O operations (two per block for the last 50 blocks, plus one for the added block). In the worst case, the file will need to be moved, requiring a total of 201 disk I/O operations. With linked allocation, the added block may be put anywhere on the disk, and its pointer set to point to the 51st block. Similarly, the pointer of the 50th block must be changed to point to the added block. This requires 50 blocks to be read (to locate the 51st block), and two blocks to be written (to adjust the pointer of the 50th block and to write the added block), for a total of 52 disk I/O operations. With indexed allocation, since all addresses of the blocks fit into a single index block, that block must be read, updated, and written, and the added block must also be written, for a total of 3 disk I/O operations.
   (c) Only 1 block need be written, so there will be one disk I/O operation to do so. In the worst case, the file will need to be moved, requiring a total of 201 disk I/O operations. With linked allocation, the added block may be put anywhere on the disk, and the pointer of the last block in the file set to point to it. This requires 100 blocks to be read (to locate the 100th block), and two blocks to be written (to adjust the pointer of the 100th block and to write the added block), for a total of 102 disk I/O operations. With indexed allocation, since all addresses of the blocks fit into a single index block, that block must be read, updated, and written, and the added block must also be written, for a total of 3 disk I/O operations.