EECE.4810/EECE.5730: Operating Systems

Spring 2017

Final Exam Solution

1. (*16 points)* ***Storage devices***

Assume you have a magnetic hard disk with tracks numbered from 0-127, track 0 being the outermost track. As a simplifying assumption, assume all tracks have 64 sectors numbered 0-63. Given the list of read requests to specific sectors listed below, show the order in which the sectors would be read using (a) shortest seek time first (SSTF) scheduling and (b) SCAN scheduling. Use the following assumptions:

- *Given a sector, S, another sector in the same track as S will be closer (and therefore have a shorter seek time) than a sector in a different track.*
- *Seek time is directly proportional to the difference between track numbers. For example, moving from track 1 to track 2 would take half as long as moving from track 5 to track 7.*
- *Multiple sectors in the same track will be read in ascending order (lowest number first).*
- *The sequence should always start with the first request in the list (track 40, sector 5). Once that request has been serviced, all other requests can be serviced in any order.*

Request list:

i. *Track 40, sector 5*
ii. *Track 15, sector 60*
iii. *Track 100, sector 0*
iv. *Track 95, sector 12*
v. *Track 40, sector 63*
vi. *Track 0, sector 0*
vii. *Track 15, sector 59*
viii. *Track 127, sector 63*

SOLUTION: The table below shows the ordering of requests for each of the two algorithms. The SCAN solution assumes the disk head starts by moving toward the center of the disk, toward higher track numbers. If you assume it starts by moving out, then the requests from track 15 are serviced first (once it completes servicing track 40), and the order will be the same as SSTF.

	SSTF	SCAN
1	(i) T40, S5	(i) T40, S5
2	(v) T40, S63	(v) T40, S63
3	(vii) T15, S59	(iv) T95, S12
4	(ii) T15, S60	(iii) T100, S0
5	(vi) T0, S0	(viii) T127, S63
6	(iv) T95, S12	(vii) T15, S59
7	(iii) T100, S0	(ii) T15, S60
8	(viii) T127, S63	(vi) T0, S0

2. *(18 + **6** points)* ***File system implementation***

a. *(6 points) Operating systems often contain direct support for one specific file type but otherwise require applications to impose other types based on file structure. What one type must the operating system support, and why?*

SOLUTION: Operating systems must support executable files because loading and running a program requires intervention from the kernel.

b. *(6 points) The open-file table, which contains information about open files, is typically split into one centralized table and several per-process tables. Name one piece of information stored in the centralized table and explain why it is kept centrally, and name one piece of information stored in a per-process table and explain why it is kept on a per-process basis.*

SOLUTION: We discussed the following information in class and which table (central or per-process) it is kept in. One thing from each list is sufficient for this problem.

- Central table
  - File open count (# processes accessing file)—when this number reaches 0, file can be closed. A central copy is more space efficient and easier to synchronize access to than multiple per-process copies.
  - Disk location of the file—the physical file location. Since every file access must go through the file system anyway, no need to keep multiple copies of this info.
- Per-process table
  - File pointer—location at which current process may read/write file. May be different for every process.
  - Access rights—allowed operations on this file for this process. Again, may be different for every process.

c. *(6 points) A file created on a Unix system should be readable to the public, readable and executable to its group, and readable, writeable, and executable to its owner. What 3-digit integer represents the access rights for this file? Briefly explain your answer.*

SOLUTION: When specifying Unix access rights using a single integer, each digit is a three bit value in which the leftmost bit enables read access (if it's set to 1), the middle bit enables write access, and the rightmost bit enables execute access. The digits are ordered as (left to right) owner, group, and public. Therefore, the access rights should be written as follows:

- Owner: R W X → $111_2 = 7$
- Group: R X → $101_2 = 5$
- Public: R → $100_2 = 4$

Therefore, the three-digit integer representing this file's access rights is 754.

1 (continued)

d. *(6 points, **EECE.5730 only**) We discussed the different layers of a typical file system, from the logical file system down to the device drivers. Some of these layers deal with logical addresses within a file, while others deal with physical addresses. Explain how and when logical file addresses and physical file addresses are used.*

SOLUTION: Logical file addresses represent an offset within the file—as a logical unit, a file is seen as a contiguous collection of blocks, starting with block 0. The application and logical file system use these addresses.

Physical file addresses represent an actual block location on disk. Multiple blocks within the same file may have contiguous addresses or they may be scattered, depending on the block allocation scheme used. The basic file system and I/O control levels of the file system use these addresses.

*3. (18 + **9** points) **Block allocation and free space management***

a. (6 points) Explain the difference between extent-based file allocation and linked file allocation.

SOLUTION: Extent-based file allocation is a form of contiguous allocation in which a file is broken into multiple contiguous chunks (extents); linked file allocation typically allocates space in units of one block, although blocks can be clustered together for performance, and builds a linked list of blocks.

The biggest difference is in how the addresses are stored. In an extent-based system, the file system stores the starting address and size of each extent as part of the file's metadata. In a linked file allocation system, the file system stores the address of the first file block as part of the metadata, and each block stores the address of the block that follows it in the linked list.

b. (6 points) Explain the difference between basic linked allocation of file blocks and the file allocation table (FAT) block allocation scheme.

SOLUTION: Both schemes maintain a linked list of file blocks. In basic linked allocation, each block contains the address of the block that follows it in the linked list. In a FAT block allocation scheme, the linked list is stored in a table (the FAT) within the file metadata.

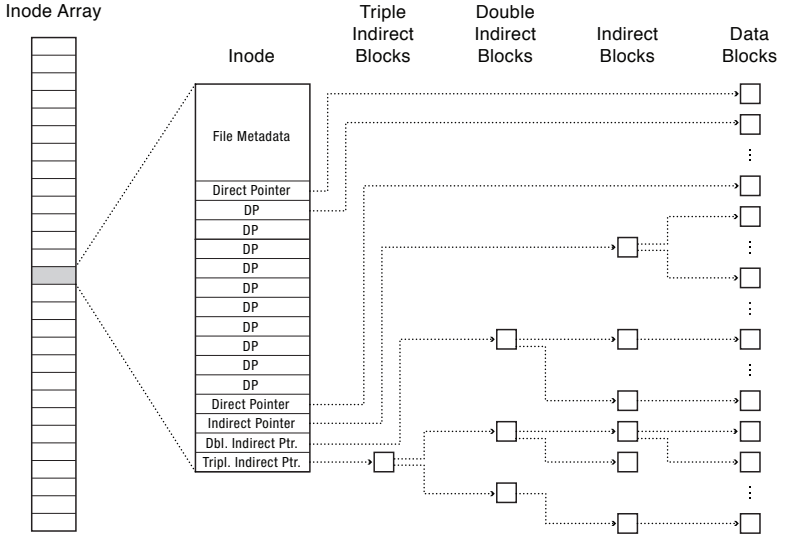
c. (6 points) The two most common methods for managing free space in a file system are bitmaps and linked free space lists. Give one benefit of a bitmap over a linked free space list, and one benefit of a linked free space list over a bitmap.

SOLUTION: Bitmaps make it relatively simple to find free blocks, especially when contiguous groups of blocks are requested. A linked free space list is much more space efficient than a bitmap.

3 (continued)

*d. (9 points, **EECE.5730 only**) Recall our discussion of the Berkeley Fast File System (FFS), which organizes its inodes as shown below. Each inode contains 12 direct pointers to disk blocks, as well as three other pointers—an indirect pointer, a doubly indirect pointer, and a triply indirect pointer. (Only the inode array and a single inode are shown—not all pointers.)*

*Assume each pointer is 32 bits and each disk block is 8 KB. If a file in this file system is 16 GB, how much space is allocated to store all of the pointers (whether direct or indirect) required to access all blocks in that file? **Show all of your work for full credit.***



SOLUTION: What the original picture didn't show is how the pointers are used—each pointer goes to a single 8 KB disk block. The first 12 blocks—those accessed by the direct pointers contain data, while the other blocks contain more pointers to more blocks. The number of additional levels of block pointers depends on the level of indirection. (Full picture above)

To determine the amount of space required to store the block pointers, consider the following:

- The file is 16 GB = 2^{34} bytes, while each block is 8 KB = 2^{13} bytes.
- So, the file is composed of $2^{34} / 2^{13} = 2^{21}$ blocks, each of which requires its own pointer.
- Each indirect block contains $2^{13} / 2^{2} = 2^{11}$ pointers, since a pointer is $4 = 2^{2}$ bytes.
- The necessary pointers are organized as follows:
    - Direct pointers account for 12 blocks
    - The indirect pointer points to one indirect block, which points to 2^{11} blocks.
    - That leaves $2^{21} – 2^{11}$ blocks unaccounted for (the 12 blocks handled by the direct pointers are insignificant in this calculation)
    - $2^{21} – 2^{11} = 2^{11} * (2^{10} – 1)$ → since each indirect block points to 2^{11} blocks, we need $2^{10} – 1$ of the indirect blocks that the doubly-indirect block points to.
    - The triply-indirect pointer is unused (but still takes up space)

So, the total amount of pointer space is the sum of the space in the inode, indirect block, doubly-indirect block, and $2^{10} – 1$ indirect blocks pointed to by the doubly-indirect block:

$(15 * 4 \text{ bytes}) + (8 \text{ KB}) + (8 \text{ KB}) + ((2^{10} – 1) * 8 \text{ KB} = 60 + 2^{13} + 2^{13} + 2^{23} – 2^{13}$

$= 2^{23} + 2^{13} + 60 \text{ bytes} = 8396860 \text{ bytes}$

(12 points) ***Reliability***

a. *(6 points) Explain why methods for ensuring reliability in file systems center on operations that write a single sector.*

SOLUTION: Writing a single sector is the only atomic operation disks support, so methods for reliability in file systems—preventing loss of data—aim to make multi-step changes to the file system reliant on that one atomic operation.

b. *(6 points) A Linux variant called TxOS, developed at the University of Texas, supports transactions with shadowing by decomposing inodes into two parts: a header that contains infrequently modified data about each file, and a data component holding fields that are commonly modified by system calls. The header contains a pointer to the related data component, and the data component contains a pointer to the header.*

   Explain how this inode organization makes it relatively easy to implement shadowing for changes to a file's metadata.

SOLUTION: Shadowing requires the file system to create a "shadow copy" of any data to be changed while maintaining a pointer to the current version. Changes are made to the shadow copy and committed by changing the pointer from the current version to the shadow copy.

By splitting the metadata into a header and data section, creating a shadow copy of the frequently changed data without copying the header is simple. The pointer within the header section can be pointed to the shadow copy of the data once changes to it are complete.

4. *(21 points)* ***Distributed systems***

a. *(6 points) In a distributed system, a process running on one node sends a byte stream to another node; the byte stream is split into several messages. How will the system recognize and handle (i) messages received out of order and (ii) dropped messages? In your answer, specify which process (sender or receiver) is responsible for detecting each of these issues.*

SOLUTION: (i) The header of each message contains a sequence number. The receiving process can use this number to place messages in the correct order.

(ii) The sending process waits for acknowledgement that each message has been received. If that acknowledgement is not received within a predefined amount of time, the sending process assumes the message was dropped and retransmits it.

b. *(6 points) Distributed file systems often allow nodes to cache local copies of shared data, using either write-through or write-back policies to handle changes to those data. List one benefit of using write-through file caching, and one benefit of using write-back file caching.*

SOLUTION: Write-through caching provides good reliability, as every change to a cached copy is written to the permanent state of the file.

Write-back caching provides better performance, as multiple repeated writes are significantly slower than only writing data back when strictly necessary.

c. *(9 points) A client process uses a binary search tree to store a set of 32-bit integers, one integer per node. The 32-bit address of the tree's root node is passed as the only argument to a remote procedure call to be executed in a server process.*

If the binary search tree contains 15 values, what is the minimum amount of space required to marshal the parameter or parameters of this function such that the remote procedure can access the entire binary search tree stored by the client? Show your work for full credit.

SOLUTION: The amount of space required for the binary search tree is dependent on the organization of data within it, but a complete binary tree—one in which every level is full—can be efficiently stored in an array, with the root of the tree in the first array location, its children in the next two locations, and so on. The number of nodes should be passed at the beginning of the array so the tree can be unpacked at the server. A sample array organization is shown below.

15	8	4	12	2	6	10	14	1	3	5	7	9	11	13	15
# nodes	root	children of root		children of 4		children of 12		children of 2		children of 6		children of 10		children of 14	

Each integer in the array takes up 4 bytes, while the number of nodes can potentially be stored in just a single byte. The minimum amount of space required to store this tree is therefore:

(15 nodes) * (4 bytes/node) + (1 byte for # nodes) = 60 + 1 = **61 bytes**

*5. (15 + **5** points)* ***Protection and security***

a. (9 points) Given the access matrix below, answer questions (i)-(iii).

Domains	**Objects**						
	F_1	F_2	F_3	D_1	D_2	D_3	D_4
D_1	read* write*		write		control		
D_2		owner					
D_3	read owner		read*				
D_4		write	write			control	

i. Can a process in domain D_1 grant read privileges for object F_3 to a process in domain D_2? Why or why not?

SOLUTION: Yes—D_1 has control rights for D_2.

ii. Can a process in domain D_3 revoke the write privileges of a process in domain D_1 for object F_3? Why or why not?

SOLUTION: No—D_3 would need owner rights for F_3 to revoke privileges in another domain.

iii. Can a process in domain D_2 give read, write, and execute privileges for object F_2 to a process in domain D_3? Why or why not?

SOLUTION: Yes—D_2 has owner rights for F_2.

6 (continued)

b. *(6 points) Two professors are discussing the security of a distributed system in which the key used to encrypt messages is known. Professor A claims that knowing the encryption key will allow him to decrypt any message sent between nodes in the system once he determines the type of cryptography used, thus making the system insecure. Professor B claims it is possible for the system to be secure, depending on what method is used to encrypt messages. Which professor is right, and why?*

SOLUTION: Professor B is correct. Asymmetric encryption schemes use two different keys—a public key for encryption and private keys for decryption. Since the type of encryption used is not specified, it's possible the system is using asymmetric encryption and is therefore secure.

c. *(5 points, **EECE.5730 only**) Briefly explain how default access rights are implemented in an access list.*

SOLUTION: An access list is a per-object collection of access rights, organized as pairs of domains and their associated access rights. A "default" domain can be added to the access list—any domain that is not explicitly listed in the access list will use the rights associated with the default domain.