Q1	Q2	Q3	Q4	Q5	Tot

CEng 334 - Introduction to Operating Systems

Spring 2018-2019, Final, May 23^{rd}, 2019
(6 pages, 5 questions, 104 points, 120 minutes)

METU Honor Code and Pledge

METU Honor Code: The members of the METU community are reliable, responsible and Honorable people who embrace only the success and recognition they deserve, and act with integrity in their use, evaluation and presentation of facts , data and documents.

I have read and understood the implications of the METU Honor Code. To be precise, I understand that this is a ***closed*** *notes exam, and I am forbidden to access any other source of information other than provided within the exam. I will TURN OFF all my electronic equipment (phones, smart watches, etc.) and put it off the table along with other notes and materials that I may have with me. I understand that leaving electronic devices on during the exam is strictly forbidden. I understand and accept to obey all the rules announced by the course staff, and that failure to obey these will result in disciplinary action.*

Name: _____ **No:** _____ **Signature:** _____

QUESTION 1.(15 points)

a) What is the difference between a port and a bus?

Only one device can be connected to a port. Multiple devices can connect to a bus.

b) In memory-mapped I/O, how does the O.S. check the status of, send commands and receive/send data from a device?

The registers of the device is mapped into the memory address space of the computer and the OS can access them by reading/writing from those addresses.

c) How does the DMA relieve the load on the CPU? Explain briefly.

The DMA can copy the data between the disk and the memory without going through the registers of the CPU, and notify the OS only when the whole transfer is completed.

d) What is the difference between a character device and a block device in terms of interface? Give an example for each type of device?

A character/block device is one with which the driver communicates by sending and receiving single characters/entire data blocks. Character/block device example: keyboard/disk.

e) How does an operating system support multiple filesystem types (like FAT, UFS, ISOFS, ...)? How is an operation on a file is linked with underlying filesystem implementation?

Support for multiple filesystem is done through Virtual File System (VFS) which maps the paths to the concrete filesystem operations, which are loaded during the mount operation.

QUESTION 2.(30 points)

a) (10 pts) Assume the assistant user is compromised by an intruder to the system in the following domain access matrix. Intruder likes to give read, write permissions to all users in the system as much as possible. Taking all access rights of the assistant user, fill in the matrix with those new access rights:

Domain \ File/Domain	myfile	yourfile	ourfile	guest	student	assistant
guest	read write	read	write			
student	read, write	read	read, write*			
assistant	read,write owner	read*, write	read write		switch	

b) (20 pts) Consider a process whose virtual memory is split into 16 pages.

Virtual Page	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Content	C	C	I	U	H											S

where **C**, **I**, **U**, **H**, **S** refers to the **C**ode, **I**nitialized Variable, **U**ninitialized Variable, **H**eap, **S**tack sections respectively. When the process starts execution, it makes the following page-references:

X1,W16,R3,X2,W4,R16,X1,W3,R2,R4,X2,R3,W5,W4,R5

where **X**, **R** and **W** indicates the type of access as e**X**ecute, **R**ead and **W**rite. The number next to the access letter indicates the virtual page number that is being accessed. For instance **R16** means that the process is Reading from virtual page **16**.

Now consider a computer system with only 3 physical page frames. Given the reference pattern, how would the content of the physical memory and swap change over time if we were to use Least Recently Used (LRU) page replacement policy. At each page-fault, indicate the source of the page; **E**xecutable), **Z**ero page or **S**wap. Also if a page is evicted, indicate the destination of the page; **-** (simply discarded), **E**xecutable, **Z**ero page or **S**wap. If multiple page frames are equally likely to be evicted, pick the page frame with the lower index.

Page reference	X1	W16	R3	X2	W4	R16	X1	W3	R2	R4	X2	R3	W5	W4	R5
Page in from	**E**	Z	E	E	Z	S	E	E	E	S	-	-	Z	S	-
Frame1	1	1	1	2	2	2	1	1	1	4	4	4	5	5	5
Frame2	.	16	16	16	4	4	4	3	3	3	3	3	3	3	3
Frame3	.	.	3	3	3	16	16	16	2	2	2	2	2	4	4
Page out to	**-**	-	-	-	S	-	-	S	-	-	-	-	-	-	-

Frames correct: 7 (-0.5 pts for each incorrect answer)
Page in correct: 7 (-0.5 pts for each incorrect answer)
Page out to correct: 6 (-0.5 pts for each incorrect answer) Only Frames upto the first wrong column are considered as correct.
For incorrect Frames, page in and page out's are automatically considered as incorrect.

Name:_____ **Id:** _____

QUESTION 3.(15 points)

You are given a partition with size 128MB ($128 \cdot 1024 \cdot 1024 = 2^{27}$ bytes exactly). Assume formatting this partition with the **FAT16** filesystem where each FAT entry takes 16 bits. If your answer requires arithmetical operations like addition/subtraction, give expressions, do not calculate.

a) What is the smallest possible cluster size that utilize this disk? 2^{11} bytes

b) What is the total size of FAT for this cluster size 2^{17} bytes.

c) What is the largest possible file in this filesystem (assume only one copy of FAT, no boot block, minimum number of directories). $2^{16} - 2^6 - 1$ clusters.

d) Assume that the O.S. wrote the data blocks of a newly created file on the dist, updated the FAT table on the disk, after which the system crashed. What kind of an integrity problem will occur on the filesystem? How can a repair utility find the problem and how can it fix it?

Directory entry could not be created. Dangling FAT entry is found and recovered as a new unnamed file

e) (Bonus, 2 pts) Assume that you want to move all files under directory B to directory A and remove directory B afterwards. The easy solution is to move all files from B to directory A and them remove B. Considering that there are thousands of files under B, propose a faster and wiser solution to this specific operation. Explain very briefly.

FAT chain of B can be added at the end of A with single FAT update. Then, size of A at metadata is updated, end end of entries marker is deleted so A directory entries are followed by B naturally. B directory entry is removed from B's parent without deallocating it.

QUESTION 4.(17 points)

You are given a partition with size 128MB (128·1024·1024=2^{27} bytes exactly). Assume that the partition is formatted as Unix filesystem with block size of **4096** and using **2 bytes data block pointers**. There are 1024 inodes in total with each inode taking 128 bytes. Assume that there are 12 direct pointers, one indirect pointer, one double indirect pointer but **no triple indirect pointers**. If your answer requires addition/subtraction, give expressions, do not calculate. All answers are in 4K blocks

a) How many blocks are used for storing the inodes? 32 blocks.

b) How many blocks are used for allocation information of inode and data blocks? 1+1 blocks.

c) What is the size (in number of blocks) of the largest file in this filesystem structure if you ignore the 128MB limit? 12+2048+2048*2048 blocks.

d) When a file entry is removed from a directory, the inode of the file may still be kept. How can this be possible? Which information stored in the inode is used for not freeing the inode?

There might be hard links to this inode. Inodes link_count field is checked if it is zero.

e) Assume that the boot block, superblock and group descriptor uses only 2 blocks. Also assume that there is only root directory and no other files in the filesystem. What is the size (in number of blocks) of the largest possible file in this 128MB filesystem, considering all overhead blocks?

$2^{17} - 17 - 2 - 2 - 32 - 1 = 2^{17} - 54$ blocks

f) (Bonus, 2 pts) Assume that there are recurrent blocks in some of your files, i.e. the first 100 blocks of two files are exactly same or the last 3 blocks of most of the files are the same. In order to use storage more efficiently, you can make data pointers of two or more inodes point to the same data block. Would this allowed in the standard Unix File System? Why or why not? Propose a solution to make this possible without causing integrity issues.

A block is either allocated or not allocated in UFS. Deleting a file fill deallocate it without considering other possible references. Data block allocatin information should keep number of references as well. So instead of a bit a byte could be used to count references.

QUESTION 5.(27 points)

Three kinds of threads share access to a singly-linked list: **Searchers**, **Inserters** and **Deleters**. Searchers merely examine the list; hence they can execute concurrently with each other. Inserters add new items to the end of the list; insertions must be mutually exclusive to preclude two inserters from inserting new items concurrently. However, an insertion can proceed in parallel with any number of searches. Finally, deleters remove items from anywhere in the list. At most one deleter process can access the list at a time, and deletion must also be mutually exclusive with searches and insertions.

Write the code for Searchers, Inserters and Deleters that enforce this kind of three-way categorical mutual exclusion.

If the Searcher, Inserter and Deleter threads are denoted by **S**, **I** and **D**, then the solution should only allow the concurrent execution of **S***, **S*I** and **D**. All other possibilities such as **I I** or **I D** or **S D** should be prevented.

Solve this synchronization problem with semaphores using the template below.

```
// inserter, deleter and searcher problem template.
int nS=0, nI=0; // number of active Searchers and Inserters
mutex = semaphore(1); // mutex to prevent race on nI and nS
sD = semaphore(1); // indicates an active Deleter.
sI = semaphore(1); // indicates an active Inserter.

void Deleter(){


    doDelete();


}
void Inserter(){


    doInsert();


}
```

```
void Searcher{


    doSearch();


}
```

```
// inserter, deleter and searcher problem template.
int nS=0, nI=0; // number of active searchers and inserters
mutex = semaphore(1); // mutex to prevent race on nI and nS
sD = semaphore(1); // indicates an active deleter.
sI = semaphore(1); // indicates an active inserter.

void deleter(){
    sD.down();
    doDelete();
    sD.up();
}
void inserter(){
    sI.down(); // allow only one inserter
    mutex.down();
    if (nS) nI++;// if there are active searchers
    else{ // no searchers and no inserters
        sD.down(); // wait for the active deleter
        nI++;}
    mutex.up();
    doInsert();
    sI.up(); // allow another inserter to get in
    mutex.down();
    if (nS) nI--;
    }else{
        sD.up();
        nI--;}
    mutex.up();
}
void searcher{
    mutex.down();
    if (nI || nS) nS++; //active inserter or searcher
    else{ // no inserter active, and first searcher
        sD.down(); //wait for the active deleter
        nS++;}
    mutex.up();
    doSearch();
    mutex.down();
    nS--;
    if (!nS && !nI) //no more active inserter and searcher
        sD.up();
    mutex.up();
}
```

Grading this question with explicit bullets in gradescope was not easy. Much of the synchronization challenge was the case on searchers. Synchronization between Deleters and Inserters is trivial. The code in Deleter should be checked along with the code in Inserter an Searcher. Therefore, the fact that "Inserter is not correct" bullet is checked does not mean that Inserter is completely wrong. I've tried to use the point adjustments along with these bullets evaluate your answer taking into all of your code.. If you have done wait(sD); doDelete(); signal(sD); for deleter and such that synchronization between deleters and inserters is achieved, you've got 7 points. Counting nI and nS is not sufficient.