Lebanese University
Faculty of Sciences I

Final 2020/2021
Duration: 90 minutes

I3303 - INFO324
Operating System II

Problem I — 30 points

1. Consider a Windows FAT system where the block is indexed in 16 bits. The size of Disk is 64 GB. Determine:
   a. The number of entries of the table FAT.
   b. The size of the table FAT.
   c. The size of a block, assuming that the disk space occupied by the FAT table is not counted among the 64 GB.

2. In a Unix FS, the topo table contains 16 direct entries (which point to blocks of data), two entries with a single index level, and two entries with a double index level. 8 bytes are used to represent the index of a block. What is the maximum size of a file on disk, knowing that the size of a block is 512 bytes? justify.

Problem II — 40 Points

1. We consider a system with paginated main memory. The memory is composed of 4 frames (frames) each has a size of 32 bytes. At a given moment, the memory is empty, and 2 processes are active in the system P1 (4 pages) and P2 (2 pages). The processor sends memory requests in the following order in the format [hexadecimal logical address, Process]:

[13F, P1]
[04A, P1]
[11D, P1]
[000, P2]
[2CA, P1]
[387, P1]
[139, P2]
[12B, P1]
[000, P1]
[011, P2]

A. What is the size of the main memory (in bytes)?
B. What is the size of the virtual space in bytes, knowing that a process can have a maximum of 8 pages?
C. Indicate by diagrams the evolution of the memory, as well as the number of page faults, admitting the following page replacement strategies:
a) FIFO
b) LRU
c) Second chance.

2. We assume the existence of 4 free space holes in a contiguous allocation memory, named from A to D, as shown in the right table:

Hole	Adress	Size
A	0	25 Octet
B	500	15 Octet
C	1000	5 Octet
D	2000	4 Octet

Three successive R1, R2 and R3 requests of size 10, 3 and 16 bytes must be satisfied. What is the memory address allocated to each of these requests if the allocation strategy is:

a. First Fit?
b. Best Fit?
c. Worst Fit?
d. Next Fit

Problem III — 30 points

1. Write a program that allows you to create 100 child processes. The parent process must execute the P() function, and each of 100 children must execute the F() function. You are not required to write the code of P () and F() functions.

2. Rewrite the program of part 1, taking into account that at each moment, **a maximum of 10 child processes can be simultaneously executing the function F(),** i.e., when 10 processes are executing the function F(), any other process wishing to execute its function F() must be blocked until one of 10 processes terminates.

I3303.E 20-21

Final

Pb I (30 pts) = (20 + 10)

windows FAT

- index (block) = 16 bits
- disk size = 64 GB

a) nb of entries of FAT?

- size (disk) = 64 GB = $2^{30} \times 2^6 = 2^{36}$
- there are 2^{16} blocks

$\Rightarrow$ nb of entries = 2^{16} (7)

b) size (FAT) = 2^{16} x size(entry)
= $2^{16} \times 2^{4} = 2^{20}$ (5)

c) size (block) = $\frac{2^{36}}{2^{16}} = 2^{20} = 1$ MB (8)

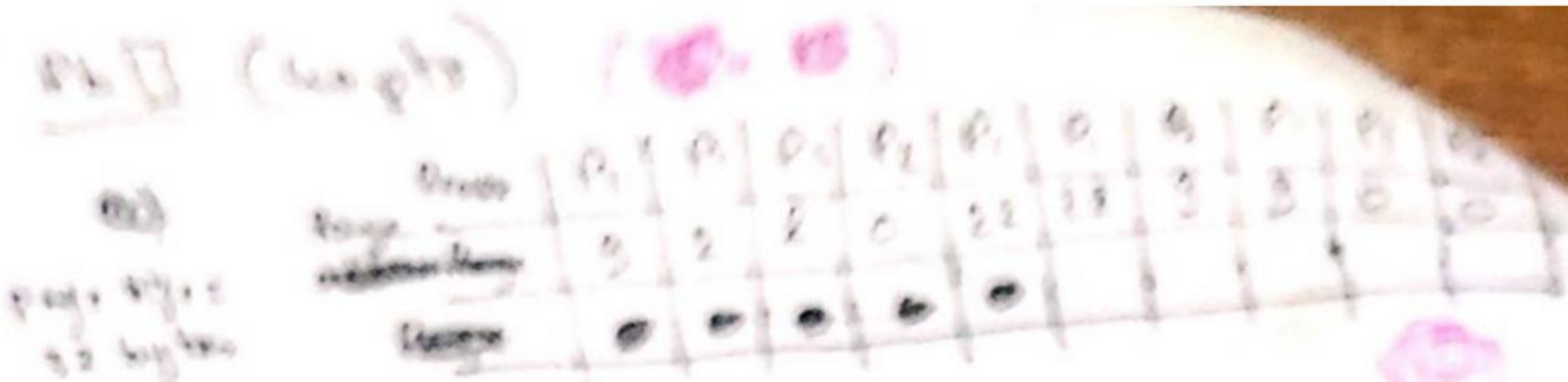
2) Unix FS (10 pts)

- block size = 512 bytes
- max size of a File

size = $16 \times 512 + 2 \times 64 \times 512$
$+ 2 \times (64)^2 \times 512$

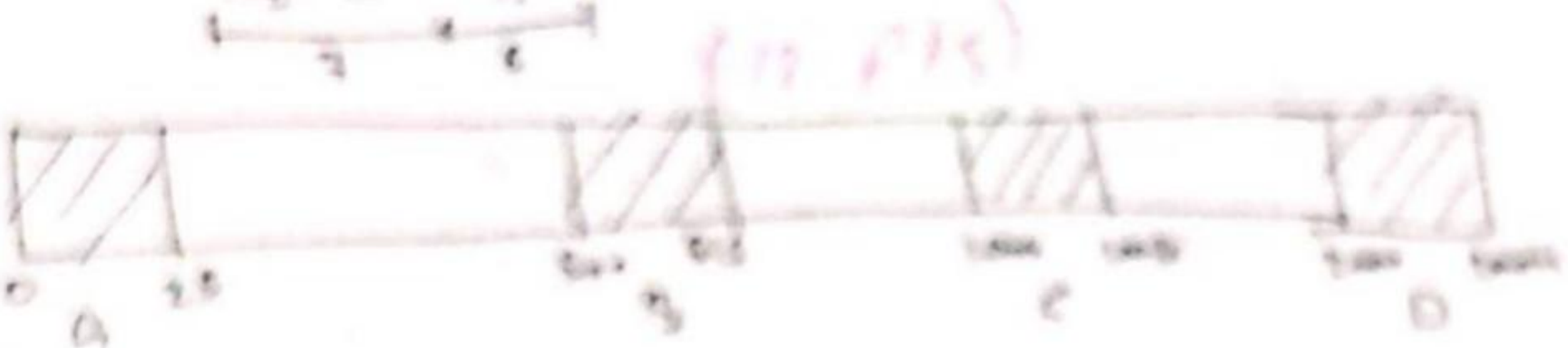
(10 pts)

(1)



b) LRU ⇒ 10 page faults

c) Second chance ⇒ 10 page faults



2)

R1 → 10
R2 → 3
R3 → 16

a) First Fit?

R1 satisfied in A
R2 satisfied in B
R3 satisfied in C

(3)

b) Best Fit

R1 → B
R2 → B or D
R3 → A

(3)

c) Worst Fit

R1 → A
R2 → A or B
R3 can't be satisfied ⇒ we should compact

(3)

d) Next Fit

R1 → B
R2 → A
R3 → compact

(3)

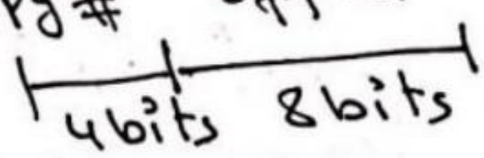
II (18 pts)

1) 4 frames memory paginated

size frame = 256 bytes = 2^8

Process	P_1	P_1	P_1	P_2	P_1	P_1	P_2	P_1	P_1	P_2
address	13F	04A	11D	000	2CA	387	139	12B	000	011
Page	1	0	1	0	2	3	1	1	0	0

virtual address 12 bits (2)



2^4 pages = 16 pages

A) size of memory = $4 \times 2^8 = 2^{10} = 1024$ bytes (2)

B) size of virtual address space = $2^4 \times 2^8 = 2^{12}$ (1)

C) FIFO, LRU and Second chance

FIFO (4)

	Page fault	Frames			
$(1,P_1)$	y	$(1,P_1)$			
$(0,P_1)$	y	$(0,P_1)$	$(1,P_1)$		
$(1,P_1)$	N	$(0,P_1)$	$(1,P_1)$		
$(0,P_2)$	y	$(0,P_2)$	$(0,P_1)$	$(1,P_1)$	
$(2,P_1)$	y	$(2,P_1)$	$(0,P_2)$	$(0,P_1)$	$(1,P_1)$
$(3,P_1)$	y	$(3,P_1)$	$(2,P_1)$	$(0,P_2)$	$(0,P_1)$
$(1,P_2)$	y	$(1,P_2)$	$(3,P_1)$	$(2,P_1)$	$(0,P_2)$
$(1,P_1)$	y	$(1,P_1)$	$(1,P_2)$	$(3,P_1)$	$(2,P_1)$
$(0,P_1)$	y	$(0,P_1)$	$(1,P_1)$	$(1,P_2)$	$(3,P_1)$
$(0,P_2)$	y	$(0,P_2)$	$(0,P_1)$	$(1,P_1)$	$(1,P_2)$

9 page faults (2)

LRU (4)

	Page fault	Frames			
		F_1	F_2	F_3	F_4
$(1,P_1)$	y	$(1,P_1)$			
$(0,P_1)$	y	$(0,P_1)$	$(1,P_1)$		
$(1,P_1)$	N	$(1,P_1)$	$(0,P_1)$		
$(0,P_2)$	y	$(0,P_2)$	$(1,P_1)$	$(0,P_1)$	
$(2,P_1)$	y	$(2,P_1)$	$(0,P_2)$	$(1,P_1)$	$(0,P_1)$
$(3,P_1)$	y	$(3,P_1)$	$(2,P_1)$	$(0,P_2)$	$(1,P_1)$
$(1,P_2)$	y	$(1,P_2)$	$(3,P_1)$	$(2,P_1)$	$(0,P_2)$
$(1,P_1)$	y	$(1,P_1)$	$(1,P_2)$	$(3,P_1)$	$(2,P_1)$
$(0,P_1)$	y	$(0,P_1)$	$(1,P_1)$	$(1,P_2)$	$(3,P_1)$
$(0,P_2)$	y	$(0,P_2)$	$(0,P_1)$	$(1,P_1)$	$(1,P_2)$

9 page faults

Second chance

	Page fault	F1	F2	F3	F4
(1, P1)	y	(1, P1)*			
(0, P1)	y	(0, P1)*	(1, P1)*		
(1, P1)	N	(0, P1)*	(1, P1)*		
(0, P2)	y	(0, P2)*	(0, P1)*	(1, P1)*	
(2, P1)	y	(2, P1)*	(0, P2)*	(0, P1)*	(1, P1)*
(3, P1)	y	(3, P1)*	(2, P1)	(0, P2)	(0, P1)
(1, P2)	y	(1, P2)*	(3, P1)	(2, P1)	(0, P2)
(1, P1)	y	(1, P1)*	(1, P2)*	(3, P1)	(2, P1)
(0, P1)	y	(0, P1)*	(1, P1)*	(1, P2)*	(3, P1)
0, P2)	y	(0, P2)*	(0, P1)	(1, P1)*	(1, P2)*

9 page faults

CIT (30 = 10 + 20)

1) #include <stdio.h>
#include <unistd.h>
void main()
{
int pid; (1)
for(int i=0; i<100; i++) (3)
{
pid = fork(); (7)
if(!pid && i<100) (1) {
// Execute F();
break; (1) }
} else if(pid && i==100) (1)
// execute P();
}

(10)

---

2) void main() {
(1) int fd[2], pid, j=0; pipe(fd); (1)
for(int i=0; i<100; i++) { (2)
pid = fork(); (1)
if(!pid && i<100) (1) {
(2) while(1) {
(1) read(fd[0], &j, sizeof(int));
(1) if(j<10) {
j++;
(2) write(fd[1], &j, sizeof(int));
// execute F();
(2) read(fd[0], &j, sizeof(int));
(1) j--;
(1) write(fd[1], &j, sizeof(int));
(3) (1) break; }}}

(20)

3

```
        break; // from loop for
    else
        // execute P();
}// end for
}// end main
```

close(fd[0]); close(fd[1]);

another solution

```
void main()
{
    int fd[2], int j=0;
    pipe(fd[2]);
    P();        close(fd[0]);
    for (int i=0; i<=9; i++)
        write(fd[1], &i, sizeof(int));

    for (int i=0; i<=99; i++)
    {
        if (!fork()) {
            read(fd[0], &j, sizeof(int));
            F();
            write(fd[1], &j, sizeof(int));
            break;
        }
    }
}
```