Lebanese University	**INFO 324**	**9 June 2016**
Faculty of Sciences1	**Operating System II**	**Duration : 2 hours**

Part I (~~28~~ 27 pts)

A) Answer the following questions with justification :

8 pts

(3) a. Explain the difference between segmentation and pagination, the relative interest of each other

(2) b. What is virtual memory? give the reasons why it is advantageous to have a virtual memory mechanism on a computer

(3) c. What is a fragmentation and how to remedy it in contiguous memory allocation

B) Consider the following C program:

8 pts

1. Draw the graph generated by calling **f(3).** (3)

```
void f(int i){
    if(i<0) exit(0);
    if(i%2==0)    fork();
    f(i-1);
    }
```

2. **Draw the graph generated by the following program** ~~(4)~~ (5)

```
void main(){
    for(int i = 0 ; i <= 3; i ++)
        f(i);
}
```

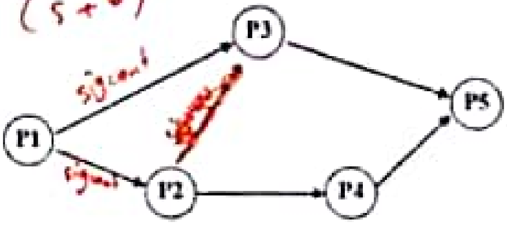
3. Add (without any modification) to the function f the required statements such that no zombie processes are generated in part 2. (2)

C) A parent process creates 5 child processes P1, P2, ..., P5. Then each child process displays its PID in respect to the order in the figure: For example the process P2 can't display its PID before P1. Also P5 can't before P3 and P4 and so on.

(11)

1. Write the program using pipes of communication.
2. Rewrite the same program using signals.

(5+6)



Part II (~~20~~ 18 pts)

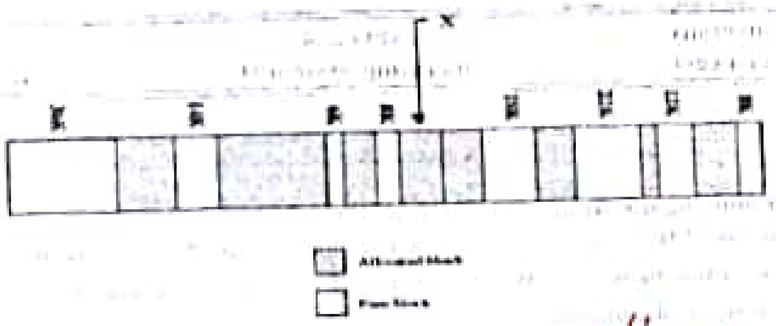
A) Consider a paginated memory with size of 48 KB and pages of size 12 KB. The following references are requested in memory: **A, B, C, D, E, B, E, F, D, A, B, C, G, F, C, B, A, B, C, F**
How many page faults are generated using the following replacement algorithms?

a. LRU (*Least Recently Used*)?
b. Second chance algorithm?

(10)

B) We wish to allocate memory space for a process of 16KB. (8 pts)

1. Simulate the functioning of First Fit and Best Fit algorithms on the following mapping.
2. What gives the Next Fit algorithm if the former allocated block is indicated by the arrow X



C) We consider the following table of segments for a process P1 (~~8~~ 4 pts)

Segment	Base	Limite
0	540	234
1	1254	128
2	54	328
3	2048	1024
4	976	200

1) Calculate the real addresses corresponding to the following virtual addresses (you may report addressing errors): (0:128), (1:100), (2:465), (3:888), (4:100), (4:344)

2) Is the virtual address (4,200) valid?

Note: the format of address is (segment#, offset)

Part III (25 pts)

A) Suppose that a disk drive has 10,000 cylinders, numbered 0 to 9999. The driver is currently serving a request at cylinder 1400. The queue of pending requests is, in the order received:
100, 1200, 900, 8000, 8100, 100, 8200, 1000, 4200
Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for the following scheduling algorithms? (For the algorithms in which the head is in constant motion, indicate the direction in which you assume it is moving initially.) 8

(a) FCFS (b) SSTF (c) Scan (d) C-look

B) We consider a file system that uses i-nodes like UNIX with few modifications as follows:

- 3 fields each of 8 bits containing information about the file
- 11 directs pointers to data blocks
- One pointer to simple indirection block where the last pointer of this block make another simple indirection

Given that each block has 1 Kb of size and occupies 2 bytes. (7 pts)

a) What is the maximum size of a file in this system?

b) Describe by figure the reading of the byte number 20992 of a file stored on disk.

C) Refer to the function written in class (i.e., **file_open** , ...): (10 pts)

a. Describe (without writing code) the steps needed to open a file.

b. How many **I/O disk** request is required to perform this task

Scanned by CamScanner

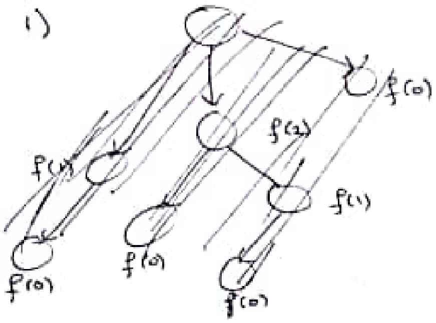
INFO 324 2015 - 2016

Part I (27 pts : A → 8 (1+2+5)
B → 8 (2+5+1)
C → 11 (5+6)

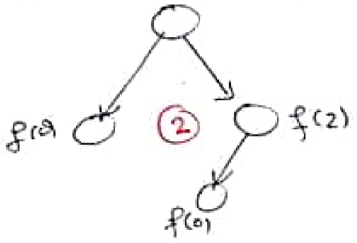
B)
```
void f(int i)
{
    if (i<0) exit(0);
    if (i%2==0) fork();
    f(i-1);
}
```

$f(3) \longrightarrow f(2) \xrightarrow{fork()} f(1) \xrightarrow{\cancel{fork()}} f(0) \xrightarrow{fork()} f(-1)$ x

1)



~~[illegible] processes~~ 4 process



2)
```
void main()
{
    for (int i=0; i<=3; i++)
        f(i);
}
```

$f(0) \xrightarrow{fork()} f(-1)$ X
$f(1) \longrightarrow f(0)$
$f(2)$
$f(3)$

⟹ ~~[illegible] processes~~

$i=0 \quad f(0) \xrightarrow{fork^1()} f(-1)$ x
$i=1 \quad f(1) \longrightarrow f(0) \xrightarrow{fork^2()} f(-1)$ x
$i=2 \quad f(2) \xrightarrow{fork^3()} f(1) \longrightarrow f(0) \xrightarrow{fork^4} f(-1)$ X
$i=3 \quad f(3) \longrightarrow f(2) \xrightarrow{fork^5} f(1) \longrightarrow f(0) \xrightarrow{fork^6} f(-1)$ X

we have 6 calls to fork, so 2^6 processes = $\boxed{64}$ 5

INFO 324 2015 - 2016

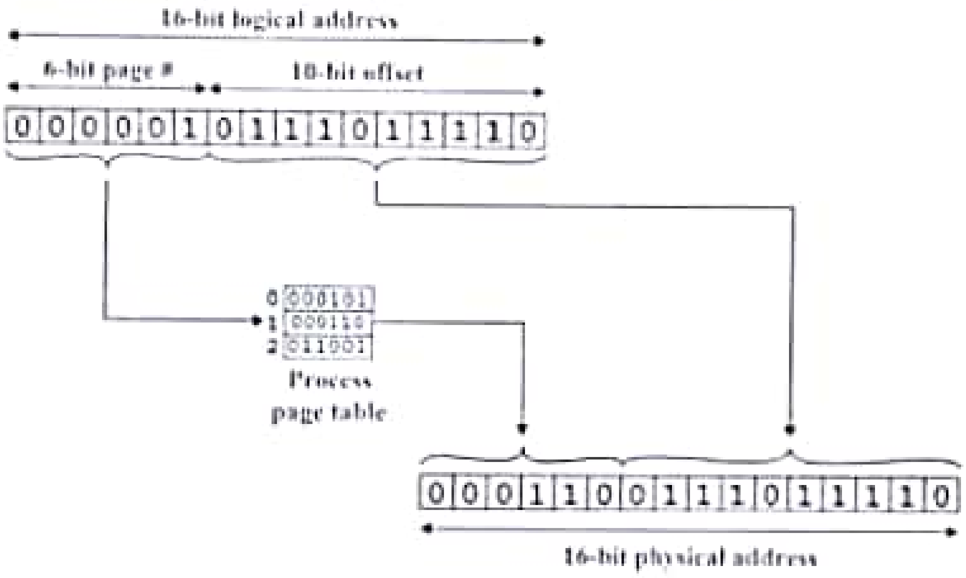
Part 1

a) Difference between Segmentation and Pagination: (3 pts)

- Paging is used to get a large linear address space without having to buy more physical memory. Segmentation allows programs and data to be broken up into logically independent address spaces and to aid sharing and protection.
- Paging does not distinguish and protect procedures and data separately.
- Segmentation distinguishes and separately protects procedures and data.
- Unlike segmentation, Paging does not facilitate sharing of procedures.
- Paging is transparent to programmers (system handles it automatically)
- Segmentation requires programmer to be aware of memory limits as programmer tries to allocate memory to functions and variables or tries to access read only memory violation, which results in segmentation fault.
- Mapping from logical to physical address is different for paging and segmentation.
  Here's an illustration based on 16 bit-address space:

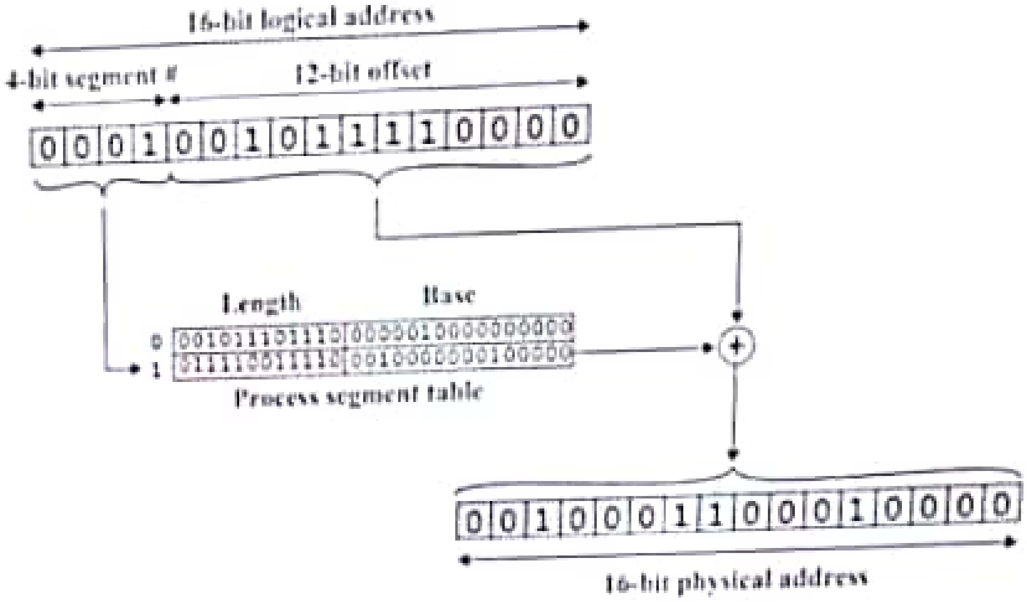
For paging:

The 6-bit page value is used to select a proper entry in process page table. the 6-bit process entry occupying the six most significant bit and the 10-bit offset occupying the 10 least significant bit forms a 16-bit physical address.



For segmentation:

The 4-bit segment of a logical address selects the proper entry in the process segment table. The base value is added to the 12 bit offset value to get the 16 bit physical address.



b) **What is virtual memory?** (2 pts)

In computing, virtual memory is a memory management technique that is implemented using both hardware and software. It maps memory addresses used by a program, called **virtual addresses**, into physical addresses in computer memory. Main storage as seen by a process or task appears as a **contiguous address space or collection of contiguous segments**. The operating system manages virtual address spaces and the assignment of real memory to virtual memory. Address translation hardware in the CPU, often referred to as a memory management unit or MMU, automatically translates virtual addresses to physical addresses. Software within the operating system may extend these capabilities to provide a virtual address space that can exceed the capacity of real memory and thus reference more memory than is physically present in the computer.

The primary benefits of virtual memory include freeing applications from having to manage a shared memory space, increased security due to memory isolation, and being able to conceptually use more memory than might be physically available, using the technique of paging.

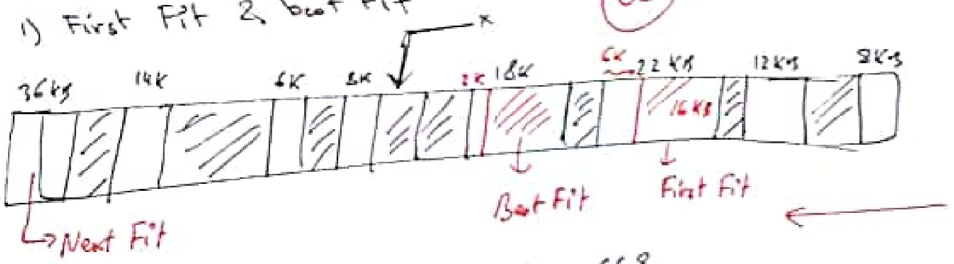
c) ~~6)~~ **What is a fragmentation?** (3 pts)

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types

B) process = 16 KB

1) First Fit 2, best Fit ω



c) 1) (0:128) → 540 + 128 = 668

① (1:100) → 1254 + 100 = 1354

(2:465) → error ω

① (3:888) → 2048 + 888 = 2936

(4:100) → 976 + 100 = 1076

① (4:344) → error

2) (4, 200) NO ①

Part III Disk : 10000 cylinders

A) - currently at cylinder 1400

queue : 100, 1200, 900, 8000, 8100, 100, 8200, 1000, 4200

8

a) FCFS : → 36400

b) SSTF → 9400

c) Scan → 9600 or 18498

d) C-look → 13400 or 16000

5300 x 7100

5300

```
3) void f (int i)
   {
     if (i<0) exit(0)
     if (i%2==0)
       {
         fork();
         wait (NULL);
       }
     f(i-1);
   }
```

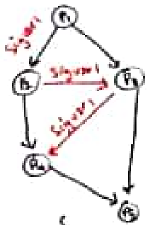
1

```
c) #include <stdio.h>
1)  #include <stdlib.h>
    #include <unistd.h>
   void main(){
   int  P12[2], P13[2], P34[2],
        ~~P35[2]~~ , P45 [2];

   int x;
   pipe(P12);  Pipe(P13); ----
   ~~close(P13[0])~~
   for (int i=1; i<5; i++)
   {   if (!fork()) break;
       if (i==1){ // P1
       {  printf(" process P1 with pid
                 %d\n", getpid());
          write (P12[1],&x, sizeof(int));
          write (P13[1], &x, sizeof(int));
       }
     else if (i==2)  //P2
     {
        read (P12[0], &x, sizeof(int));
        printf ("Process P2 ---- ");
        write (P24[1], &x, sizeof(int));
     }
     else if (i==3)  // P3

           ⋮
```

5 pts

2) signals

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
int pid, int fd[2]; int next;
  static counter=0;
void handler (int nsig)
 { ctr++
  printf(I'm the process number %d,
        with pid %d\n", ctr, getpid());
 if(read (fd[0], &next, sizeof(int))){
     [illegible]
          Kill (next, sigusr1); exit;
   [illegible] else
   }
 else
 {
  exit;
  }
 } //end handler
```



5 pts

```
void main() {
 ~~int~~  pipe(fd);
       signal (sigusr1, handler);
  for (int i=1; i≤4 ; i++)
  {
     if(!(pid=fork()))
       {  close (fd[1]);
          pause();
       }
  ~~[illegible]~~ write (fd[1], &pid, sizeof(int));
  } // end for
  printf ("I'm the process P1 with
          pid=%d\n", getpid());
  close (fd[1]);
  while (wait(NULL));
  } //end main
  read(fd[0], &next, sizeof(int));
  Kill(next, sigusr1);
  counter++;
```

Second chance Alg

Page	Fault	4 page frames			
A	yes	A			
B	yes	B	A		
C	yes	C	B	A	
D	yes	D	C	B	A
E	yes	E	D	C	B
B	No	E	D	C	B*
E	No	E*	D	C	B*
F	yes	F	E*	D	B
D	No	F	E*	D*	B
A	yes	A	F	E*	D*
B	yes	B	A	E	D
C	yes	C	B	A	E
G	yes	G	C	B	A
F	yes	F	G	C	B
C	No	F	G	C*	B
B	No	F	G	C*	B*
A	yes	A	F	C	B
B	No	A	F	C	B*
C	No	A	F	C*	B*
F	No	A	F*	C*	B*

12 page faults

Part II

A) Memory :- size 48KB } => 4 Frames
- page 12KB

a) LRU ⑤ (3+2)

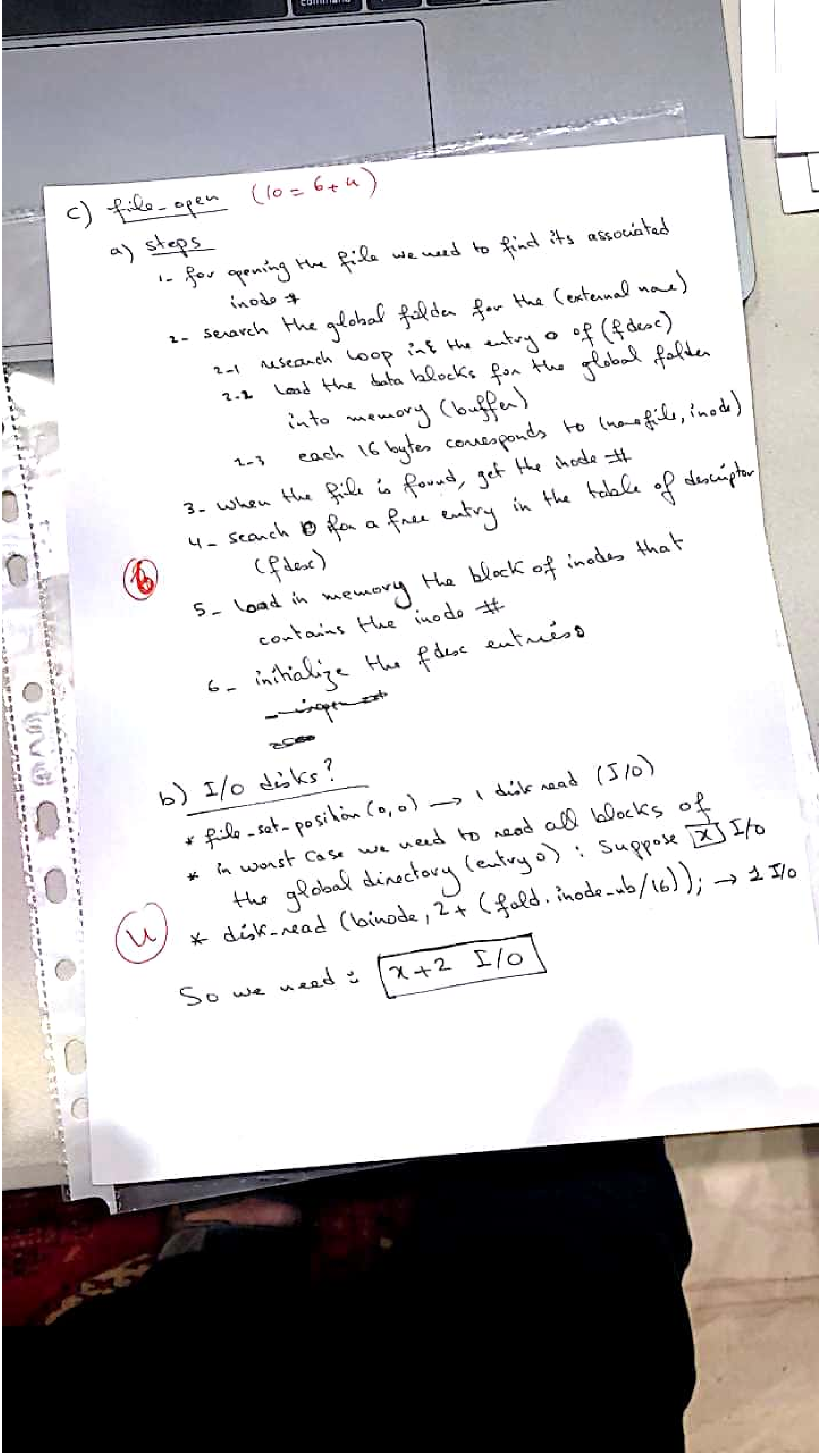
Page ref	Faults				
A	yes	A			
B	yes	A	B		
C	yes	A	B	C	
D	yes	A	B	C	D
E	yes	E	B	C	D
B	No	E	B	C	D
E	No	E	B	C	D
F	yes	E	B	F	D
D	No	E	B	F	D
A	yes	E	A	F	D
B	yes	B	A	C	D
C	yes	B	A	C	G
G	yes	B	F	C	G
F	yes	B	F	C	G
C	No	B	F	C	G
B	No	B	F	C	A
A	yes	B	F	C	A
B	No	B	F	C	A
C	No	B	F	C	A
F	No	B	F		

12 page faults

b) Second chance ⑤ (3+2)

Page ref	4 Page Frames				
	Fault	Page contents			
A	yes	A*			
B	yes	B*	A*		
C	yes	C*	B*	A*	
D	yes	D*	C*	B*	A*
E	yes	E*	D	C	B
B	No	E*	D	C	B*
E	No	E*	D	C	B*
F	yes	F*	E*	D	B
D	No	~~D*~~ F*	E*	D* E*	B
A	yes	A*	~~D*~~ F*	E*	~~E*~~ D*
B	yes	B*	A	~~D~~ F	~~F~~ E
C	yes	C*	B●	A	~~D~~ F
G	yes	G*	C*	B●	A
F	yes	F*	G●	C●	B●
C	No	F*	G●	C*	B●
B	No	F*	G●	C*	B*
A	yes	A*	F	C	~~C~~ B
B	~~yes~~ NO	~~B*~~ A*	~~A*~~ F	~~F~~ C	~~G~~ B*
C	~~yes~~ NO	~~C*~~ A*	~~B*~~ F	~~A*~~ C*	~~F~~ B*
F	No	F*	C*	B*	A*

✓ ~~14~~ page Faults
12 12 → see next page ③



c) file_open (10 = 6 + 4)

a) steps

1- for opening the file we need to find its associated inode #

2- search the global folder for the (external name)

2-1 research loop in the entry 0 of (fdesc)

2-2 Load the data blocks for the global folder into memory (buffer)

2-3 each 16 bytes corresponds to (namefile, inode)

3- when the file is found, get the inode #

4- search for a free entry in the table of descriptor (fdesc)

6

5- Load in memory the block of inodes that contains the inode #

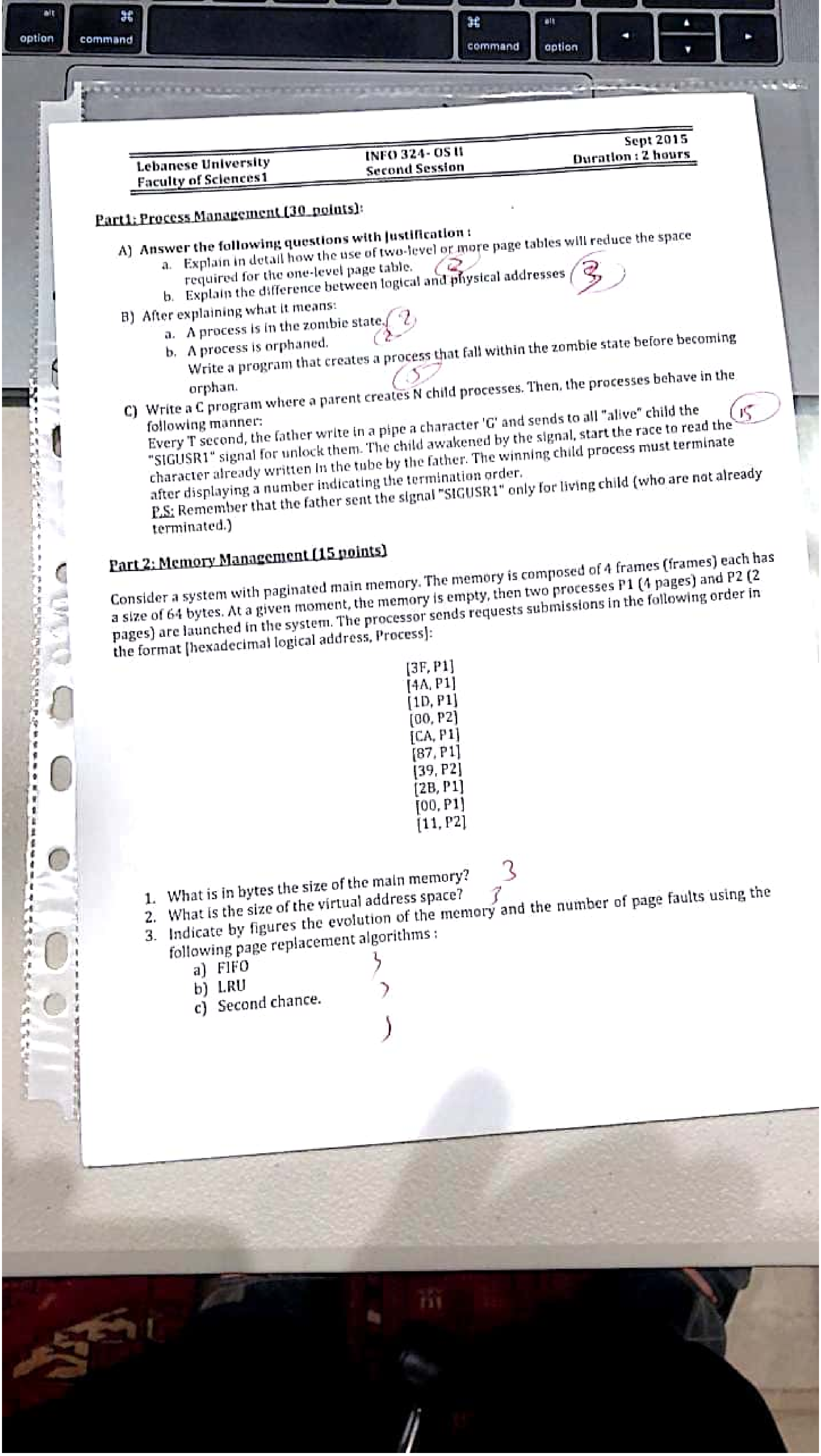
6- initialize the fdesc entries

b) I/O disks?

* file_set_position (0,0) → 1 disk read (I/O)
* in worst case we need to read all blocks of the global directory (entry 0) : Suppose $\boxed{x}$ I/O
* disk_read (binode, 2 + (fold. inode_nb/16)); → 1 I/O

4

So we need : $\boxed{x + 2 \ I/O}$



Lebanese University
Faculty of Sciences1

INFO 324- OS II
Second Session

Sept 2015
Duration : 2 hours

Part1: Process Management (30 points):

A) **Answer the following questions with justification :**
   a. Explain in detail how the use of two-level or more page tables will reduce the space required for the one-level page table. 2
   b. Explain the difference between logical and physical addresses 3

B) After explaining what it means:
   a. A process is in the zombie state. 2
   b. A process is orphaned. 2
   Write a program that creates a process that fall within the zombie state before becoming orphan. 5

C) Write a C program where a parent creates N child processes. Then, the processes behave in the following manner: 15
Every T second, the father write in a pipe a character 'G' and sends to all "alive" child the "SIGUSR1" signal for unlock them. The child awakened by the signal, start the race to read the character already written in the tube by the father. The winning child process must terminate after displaying a number indicating the termination order.
P.S: Remember that the father sent the signal "SIGUSR1" only for living child (who are not already terminated.)

Part 2: Memory Management (15 points)

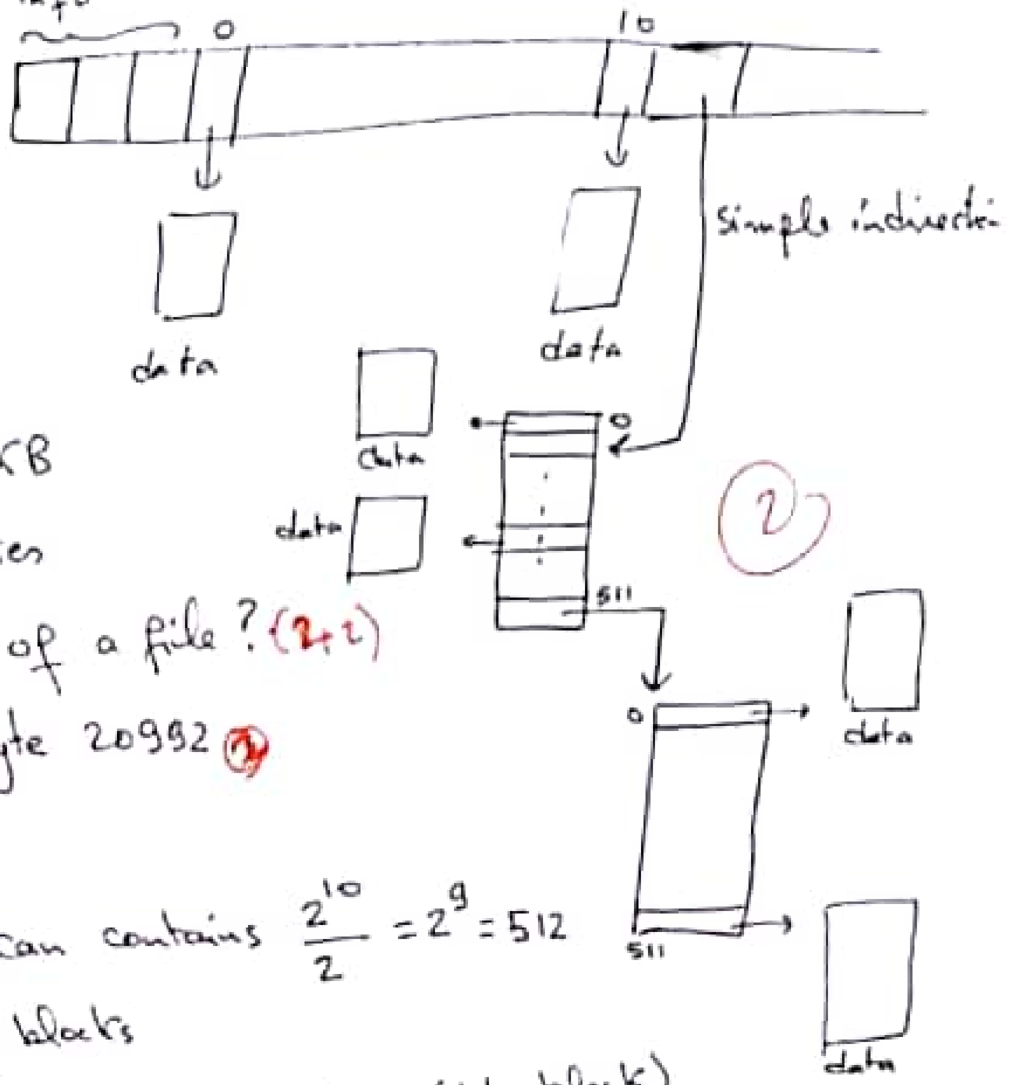
Consider a system with paginated main memory. The memory is composed of 4 frames (frames) each has a size of 64 bytes. At a given moment, the memory is empty, then two processes P1 (4 pages) and P2 (2 pages) are launched in the system. The processor sends requests submissions in the following order in the format [hexadecimal logical address, Process]:

[3F, P1]
[4A, P1]
[1D, P1]
[00, P2]
[CA, P1]
[87, P1]
[39, P2]
[2B, P1]
[00, P1]
[11, P2]

1. What is in bytes the size of the main memory? 3
2. What is the size of the virtual address space? 3
3. Indicate by figures the evolution of the memory and the number of page faults using the following page replacement algorithms :
   a) FIFO 3
   b) LRU 3
   c) Second chance. 3

Part III

B) Inode :



7 (4+3)

Block size = 1 KB

Block # = 2 bytes

a) Maximum size of a file ? (2+2)

b) reading of byte 20992 (3)

Answers :

a) each block can contains $\frac{2^{10}}{2} = 2^9 = 512$

number of blocks

Max size : 11 direct block + 511 (data block)

+ 512 (data block)

= 11 + 511 + 512 = $\boxed{1034 \text{ KB}}$ 2

11264

b) $20992 - (11 \times 1024) = 20992 - 11264 = 9728$

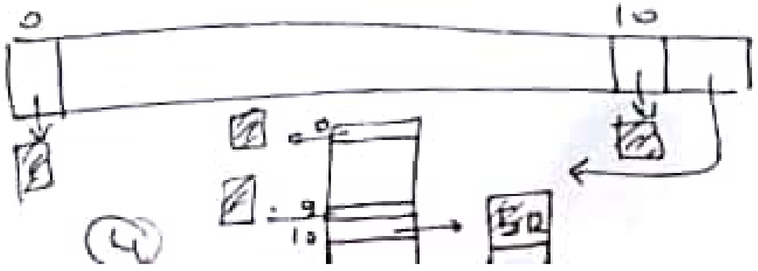
$9728 = (\underbrace{9 \times 1024}_{9216}) + 512$

So the File covers the 11 direct data block

- the first 0 ⟶ 9 (10) from the simple indirection
- In the block 10 (simple indirection), we have

offset 512

3



Part 3: File System (25 points)

A) Consider a file system with linked list and indexed allocation strategy as illustrated in the figure. The size of block is 8 KB.

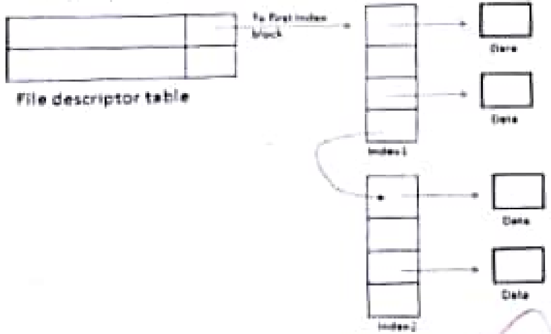


Figure: File system illustration

10

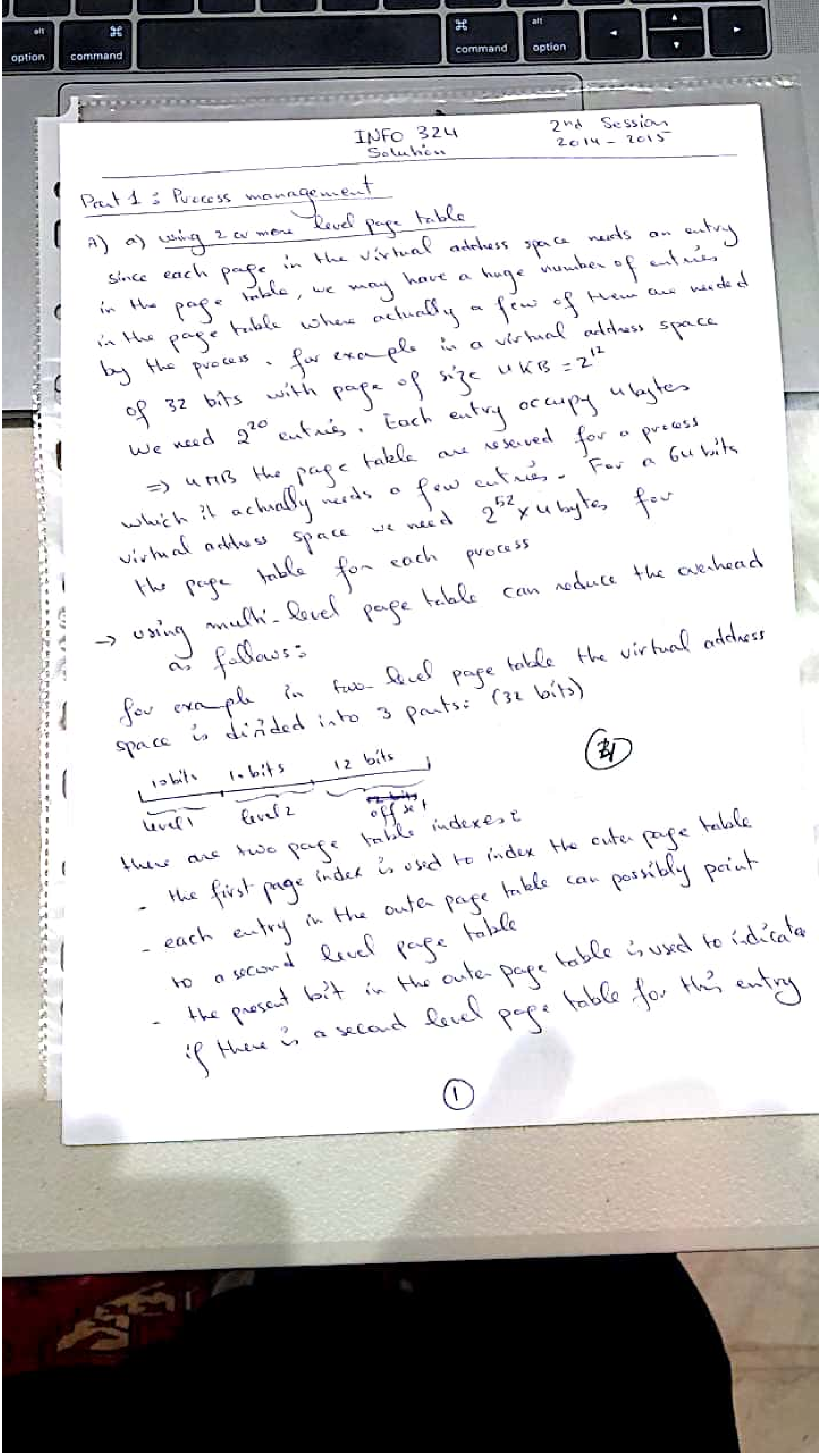
The disk blocks are numerated from 0 to n-1. Each opened file has an entry in the file descriptor table which is loaded in memory. This entry contains all attributes of the file and a pointer to an index block. The index block contains k pointers to other blocks. These pointers except the last points to data block. The last pointer points to another index block and so on as shown in figure. The pointer to block occupies 32 bits.

Question: What is the maximum size (in number of blocks) of a file in this system? Indicate the number of data blocks and index blocks

B) Given a FS where the topo table contains 10 entries, each corresponding to a single level indexing (each block points to a map). Given that each block occupies 2 kilobytes, the number of a block occupies 4 bytes and each block contains 16 inode:
Write the function ***delete_block (int lbd, int f)*** that delete the physical data block corresponding to the logical data block lbd of the file with descriptor f.
Note that if this block is not at the end of file, you must shift all the other data blocks that follow.

15

P.S: the use of the functions seen in the course is permitted if needed.



INFO 324
Solution

2nd Session
2014 - 2015

Part 1 : Process management

A) a) using 2 or more level page table

since each page in the virtual address space needs an entry in the page table, we may have a huge number of entries in the page table where actually a few of them are needed by the process. for example in a virtual address space of 32 bits with page of size 4 KB = 2^{12}

We need 2^{20} entries. Each entry occupy 4 bytes

=> 4 MB the page table are reserved for a process which it actually needs a few entries. For a 64 bits virtual address space we need $2^{52} \times 4$ bytes for the page table for each process

→ using multi-level page table can reduce the overhead as follows:

for example in two-level page table the virtual address space is divided into 3 parts: (32 bits)

2½

10 bits	10 bits	12 bits
level 1	level 2	~~12 bits~~ offset

there are two page table indexes:

- the first page index is used to index the outer page table
- each entry in the outer page table can possibly point to a second level page table
- the present bit in the outer page table is used to indicate if there is a second level page table for this entry

Part2 : Memory management (15 pts)

$P_1 \rightarrow$ 4 pages

$P_2 \rightarrow$ 2 pages

1) size of ~~memory~~:

64 x 4 = 256 Bytes = 2^8 Bytes ③

2) size of the virtual addres space

~~Hence~~ Page size = 64 By = 2^6 Bytes

$\Rightarrow$ 6 bits is needed for the offset

- The logical address is in hexadecimal (2 digits)
- each is encoded on 4 bits $\Rightarrow$ the logical address is encoded on 8 bits $\Rightarrow$ $\underbrace{\#}_{2}\underbrace{\text{offset}}_{6}$

$\Rightarrow 2^2$ pages $\Rightarrow$ 4 x 64 = 256 Bytes = $\boxed{2^8}$ ③

3)

3F ⇒	0011 1111	63	0	P1
4A ⇒	0100 1010	74	1	P1
1D ⇒	0001 1101	29	0	P1
00 ⇒	0000 0000	0	0	P2
CA ⇒	1100 1010	202	3	P1
27 ⇒	1000 0111	135	2	P1
39 ⇒	0011 1001	57	0	P2
2B	0010 1011	43	0	P1
00	0000 0000	00	0	P1
11	0001 0001	17	0	P2

$\Rightarrow$ ~~the w = {0, 1, 0, 0, 3, 2, 0, 0, 0, 0}~~

w = {(0;P1), (1;P1), (0;P1), (0;P2), (3;P1), (2;P1), (0;P2), (0;P1), (0,P1), (0,P1)}

~~4 page~~ faults

a) FIFO

	0	0	0*	0*	0	0	0*	0*	0*
0		1	1	1	1	1	1	1	1
1					3	3	3	3	3
2						2	2	2	2
3									

③

a) FIFO

Page refs	4 page frames				
	Fault?	Page contents			
(0, P1)	✓	(0,P1)			
(1, P1)	✓	(1,P1)	(0,P1)		
(0, P1)	✗	(1,P1)	(0,P1)		
(0, P2)	✓	(0,P2)	(1,P1)	(0,P1)	
(3,P1)	✓	(3,P1)	(0,P2)	(1,P1)	(0,P1)
(2, P1)	✓	(2,P1)	(3,P1)	(0,P2)	(1,P1)
(0,P2)	✗	(2,P1)	(3,P1)	(0,P2)	(1,P1)
(0,P1)	✓	(0,P1)	(2,P1)	(3,P1)	(0,P1)
(0,P1)	✗	—	—	—	—
(0,P2)	✗	—	—	—	—

6 page faults 3

b) LRU

Page refs	4 Page Frames				
	Fault	Page contents			
(0,P1)	✓	(0,P1)			
(1,P1)	✓	(1,P1)	(0,P1)		
(0,P1)	✗	(0,P1)	(1,P1)		
(0,P2)	✓	(0,P2)	(0,P1)	(1,P1)	
(3,P1)	✓	(3,P1)	(0,P2)	(0,P1)	(1,P1)
(2,P1)	✓	(2,P1)	(3,P1)	(0,P2)	(0,P1)
(0,P2)	✗	(0,P2)	(2,P1)	(3,P1)	(0,P1)
(0,P1)	✗	(0,P1)	(0,P2)	(2,P1)	(3,P1)
(0,P1)	✗	—	—	—	—
(0,P2)	✗	(0,P2)	(0,P1)	(2,P1)	(3,P1)

5 page faults 3

c) Second chance

Page refs	4 Page Frames					out
	Fault?	Page contents				
(0,P1)	✓	(0,P1)				
(1,P1)	✓	(1,P1)	(0,P1)			
(0,P1)	✗	(1,P1)	(0,P1)*			
(0,P2)	✓	(0,P2)	(1,P1)	(0,P1)*		
(3,P1)	✓	(3,P1)	(0,P2)	(1,P1)	(0,P1)*	
(2,P1)	✓	(2,P1)	(3,P1)	(0,P2)	(0,P1)	(1,P1)
(0,P2)	✗	(2,P1)	(3,P1)	(0,P2)*	(0,P1)	
(0,P1)	✗	(2,P1)	(3,P1)	(0,P2)*	(0,P1)*	
(0,P1)	✗	—	—	—	—	
(0,P2)	✗	—	—	—	—	

5 page faults

3

Scanned by CamScanner

b) logical & physical address

the logical address is used by each process in ~~each~~ its virtual address space starting from 0 to max-1 bytes depending on the architecture and it is private for each process. whereas the physical address ~~space~~ is the real address on physical memory. ~~and~~ It has two parts

2

```
two parts   |_______frame #_______|___offset___|

virtual     |_______page #________|___offset___|
```

B) a) zombie state

a zombie process is a process that has completed execution (via the exit system call) but still has an entry in the process table waiting his parent to get its status by the (wait call)

2

b) orphaned state

An orphan process is a process that is still executing, but whose parent has died

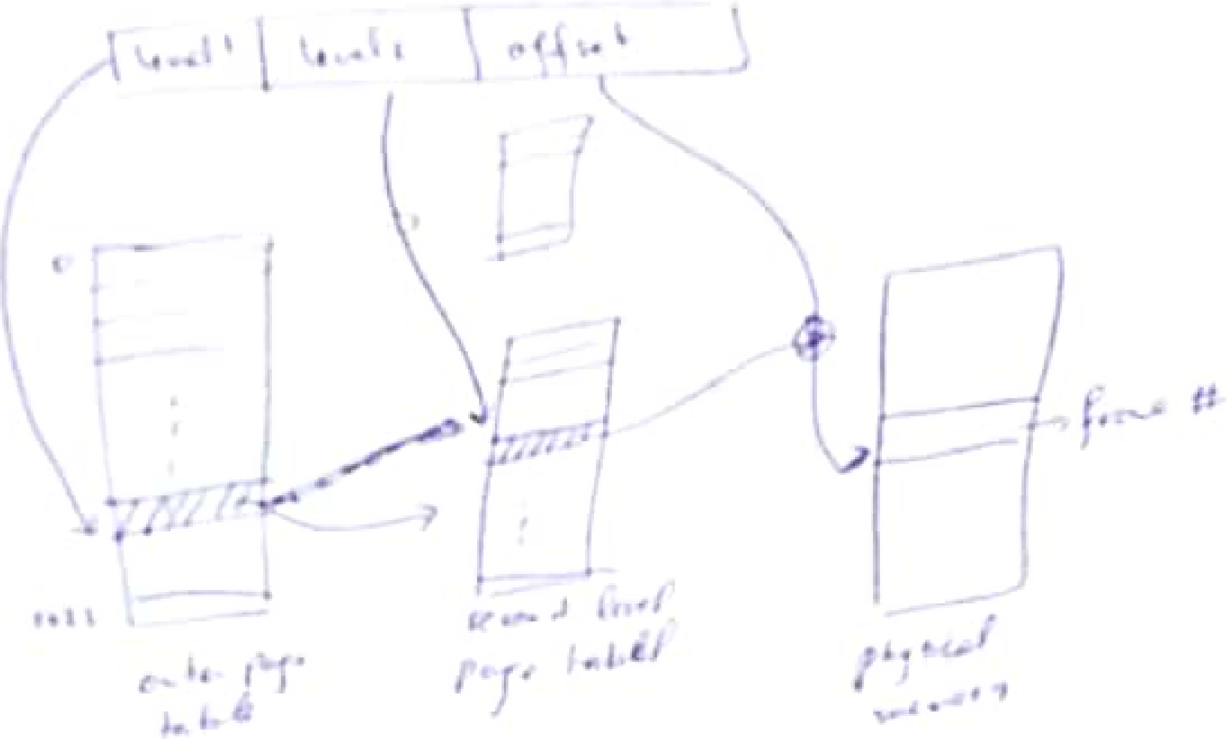
2

```
→ # include <unistd.h>
  # include <stdio.h>

  int main()
  { int pid; int i=0;
    pid = fork();
    if(!pid)
       while(i<=10) sleep(10);
    else
       exit(1);
  } return 0;
```

5

2

- the second page index is used to index the second level page table pointed to by the outer page table
- the entry of the second level page table might contain the page frame number of the desired page



For example:

for a process with 8MB of code
4MB heap
2MB stack

⇓

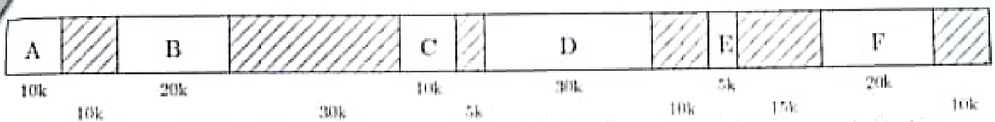
we need = 2 entries for the code
1 entry for the stack
1 entry for the heap

so we need just - the outer page table → 4×1024 bytes = 4KB
- the inner page table {for data and heap}
- the inner page table for stack

⇒ 4KB + 4KB + 4KB = $\boxed{12KB}$

while in single-level page table it needs $\boxed{4MB}$

to the process. At the instant t, the state of the main memory is described in the figure below (where A, B, C, D, E, F are processes already loaded in memory, and the other parts are free zones):



Represent the evolution of the main memory according to the arrival of each of the following successive events.

- Event 1. Arrival of program G (20k).
- Event 2. Start of program B.
- Event 3. Arrival of program H (15k).
- Event 4. Departure of the program E.
- Event 5. Arrival of program I (40k).

2. The memory of a computer contains 4-page frames and, at the beginning, all the frames are empty. How many page faults produces the following page references 3, 4, 4,1, 5, 2, 3,1,4 using, respectively, FIFO, OPTIMAL, and LRU replacement algorithms? Justify your answers by showing the contents of the frames after each reference.

Problem IV — 20 points

1. Describe (in pseudocode) the steps to be followed by the function **void Move (char * file_name, int f1, int f2)** which moves a file from a source directory (f1 is the index of its descriptor entry) to a destination directory (f2 is the index of its descriptor entry).

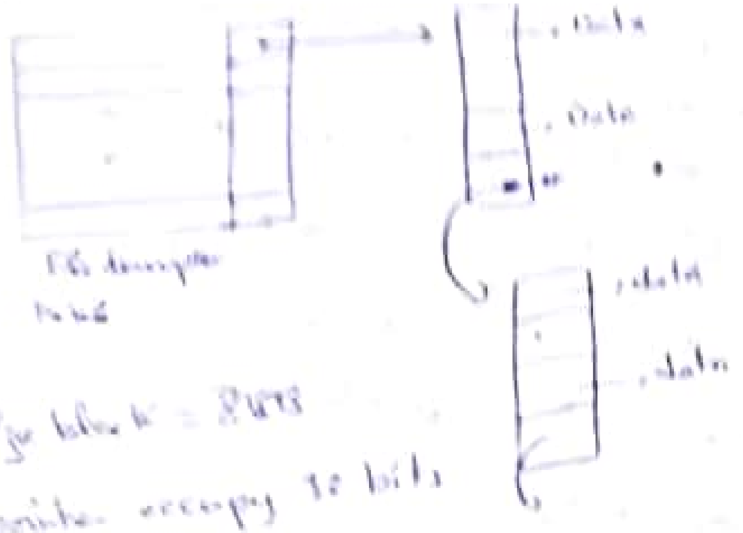
2. Describe (in pseudocode) the steps to be followed by the function **int Similar (int f, char * file_name_1, file_name_2)** which compares the contents of two files belonging to the opened directory with entry descriptor f. If both files have the same content, the function returns 1, and 0 if not.

P.S: Feel free to use the structures and functions seen in the course.

Bon travail

Part 3: File system

A) Linked list and indexed allocation strategy



10

- size block = 8KB
- pointer occupy 32 bits

Question : Max size (# blocks) of a file ?

0 → n-1 blocks => n blocks

- each block can containing $8KB/4B$ = 2K entries

~~data : (K-1) x 8KB~~

~~Max file size~~ ~~index : (n-1) * 4 bytes / 8KB~~

- each index block points to 2K-1 data blocks = 2047
- pointer occupies 32 bits => 2^{32} blocks on the system
- if m is the max number of index blocks and all these blocks are used by the file (index + data) then we have:

$$2^{32} = \underbrace{m}_{index} + (\underbrace{2047 \times m}_{data}) \Rightarrow m = \frac{2^{32}}{2048} \Rightarrow \boxed{m = 2^{21}}$$

=> the number of data block = $2^{21} \times 2047$

=> Max file size is : 2^{32} blocks → 2^{21} index block → $2^{21} \times 2047$ data blocks

```
else   // the block to delete is inside a map block but not
                          [illegible] shift all entries
B)  void delete_block (int F, int log_b_d)
    {   ~~// if the file is short, so~~
      // the file is long, so we should know in which map
      // block is the block to delete.
      // if it is in the last so no problem, we load the
      //   map block and shift all entries back one
      // if not => we should shift all entries in the map
      // block containing the block to delete and all
      // other following map blocks

    int nb_of_map_blocks=0;
    int map_of_block_to_delete;    int remaining_map_blocks=0;
    nb_of_map blocks = fdesc[F].lg%512==0 ? fdes[F].lg/512
                                 : (fdesc[F].lg/512)+1;

    map_of_block_to_delete = log_b_d /512;
    if (map_of_block_to_delete == (nb_of_map_block)-1)
     {  //the block to delete is on the last map block of the file
        // so, just shift the entries of this map block
        load_i(F, map_of_block_to_delete);
        x = log_b_d % 512;   fdesc[F].pbd = fdesc[F].map[x];
        for (int i=x+1; i<512; i++)
              map [i-1] = map [i];
        block_release (fdesc [F]. pbd);
     }
```

15 pts

```
else    // the block to delete is inside a map block but not
  {     // the last one, so we should shift all entries
        // of the fallowing map block one back
    remaining-map-blocks = nb-of-map-blocks - (map-of-block-to-delete)-1;
    x = log-b-d % 512;
    for ( i=map-of-block to-delete; i<nb-of-map-block ; i++)
    {
      for (j=x+1; j<((next-map-block*512) + 512-x); j++)
      {
        if (j%512 == 0) load-map(F, j);
        ~~map[j-1] = map[j]~~
        map [(j-1)%512] = map [j%512];
      } //end for
    } //end for
  } end else
} //end function
```

Lebanese University
Faculty of Sciences I

2nd session 2017/2018
Duration: 2 hours

I3303 / INFO324
Operating System II

Problem I

18 points

In each of the following programs, it is assumed that the parent process has Pid = 100, and that there are no other active processes in the system than those created by the program. Give all possible display results obtained by running each of the following programs.

Program 1

```
void main(){
      fork();
      if(fork())
          printf("%d\n",fork());
}
```

5

Program 2

```
void main(){
      int x;
      x =  (fork() + fork()) * fork();
      printf("%d\n",x);
}
```

8

Program 3

```
void main(){
      int p[2], x = 0, y;
      if(fork()){
            pipe(p);
            x = getpid();
            write(p[1],&x,sizeof(int));
      }
      else{
            read(p[0],&y,sizeof(int));
            printf("%d %d",x,y);
      }
}
```

5

Problem II

16 points

6 1. Using the SIGALRM signal, write a program that draws a number between 1 and 100 after one second and displays it.

10 2. Write a C program under UNIX where the parent process creates 100 child processes as follows: The father creates the first process and waits for the child to display its PID, before creating the second child. Then he waits for the second to display his PID before creating the third child, and so on.

Problem III

16 points

1. We consider a memory managed by contiguous allocation. The allocation of processes is made according to the first fit algorithm. That is, the first encountered area whose size is greater than or equal to the size of the process to be loaded is the one that is allocated

Problem II :

a)

```
void handle(int nsig) { } ②
void main() {
    srand(time(0));
  ① signal( SIGALRM, handle);
①① alarm(1);
    pause();
  ① int num = (rand() % 100) +1;
  ① printf("%d\n", num);
  }
```

⑥

b)

```
void handler(int nsig){
    printf("I'm the parent and I got a message from my child!");
 ③  signal(SIGUSR1, handler);
  }
  void main(){
  ① signal(SIGUSR1, handler);
  ① for(int i=0; i<100; i++) {
          if(fork()) pause(); ②
          else {
                                         ①
     ③}     printf("I'm the child with rank %d and pid=%d\n", i, getpid());
            kill(SIGUSR1, getppid()); break;
                                    ①
          } // end if   ①
      } // End for
  } // End main
```

⑩ { 7 x, 3 }

A	B	G	C	D	H	F	I	▨
10	20	20	10	30	15	20	40	10

2) w = { 3, 4, 4, 1, 5, 2, 3, 1, 4 }

FIFO ④ LRU ④ optimal

FIFO

ref	Fault	A	B	C	D
3	y	3			
4	y	3	4		
4	N	3	4		
1	y	3	4	1	
5	y	3	4	1	5
2	y	2	4	1	5
3	y	2	3	1	5
1	N	2	3	1	5
4	y	2	3	4	5

7 page faults

LRU

ref	Fault				
3	y	3			
4	y	4	3		
4	N	4	3		
1	y	1	4	3	
5	y	5	1	4	3
2	y	2	5	1	4
3	y	3	2	5	1
1	N	1	3	2	5
4	y	4	1	3	2

7 page faults

optimal ③

3	4	4	1	5	2	3	1	4
3	3	3	3	3	3	3	3	4
	4	4	4	4	2	2	2	2
			1	1	1	1	1	1
				5	5	5	5	5

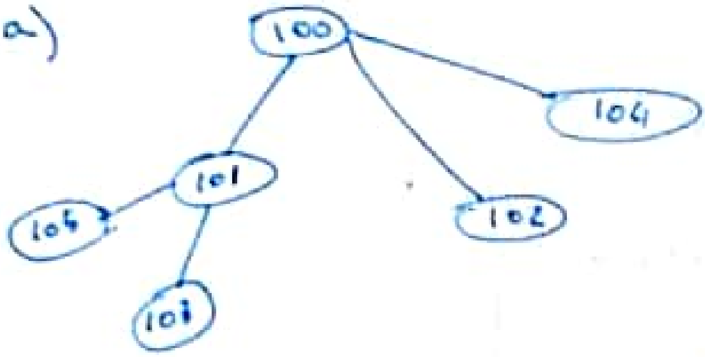
②

6 page faults

I3303/324 Second session 2018

Problem 1

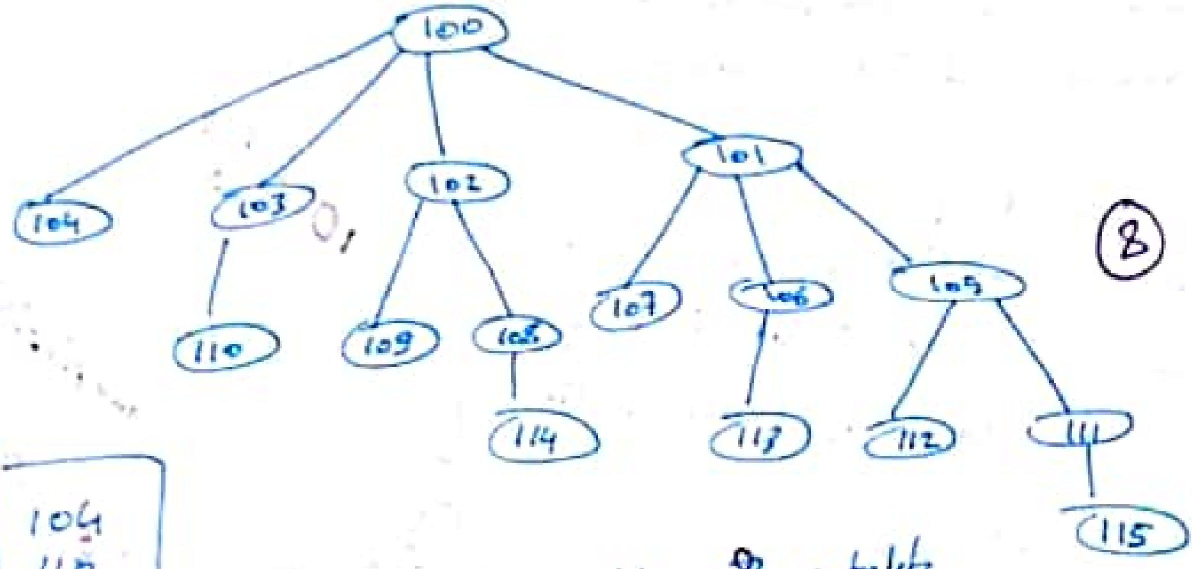
a)



5

output: 104 } pid of last 2 childs created
105

b)



8

104
110
109
112
113
114
115
107

8 outputs

other 8 outputs prints $\boxed{0}$ (child)

c) the output is 0 0

5

Lebanese University Faculty of Sciences1	INFO 324 Operating System II	17 June 2015 Duration : 2 hours

Part1 (20 minutes - 15 points)

A) Answer the following questions with justification :

a) What is the advantage of contiguous file allocation policy **vs** linked allocation policy? 2

b) Describe in detail the problem of fragmentation and how to manage it? 4

B) Given the following program

```
#include <stdio.h>
#include <unistd.h>
int main() {
int i;
for (i=0;i<3;i++)
        if (fork()) i++;
while(1);
return i;}
```

5

How many processes does this program generate? Draw the generated graph.

C) How many processes are generated by the following code :

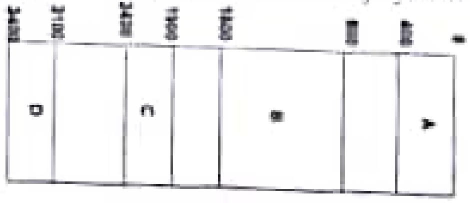
```
int main () {
  while (fork())
    execv (path, com);
  return 0;
}
```

4

Where **path** is the path to the executable and **com** is the executable process.

Part 2: Memory Management (60 minutes - 30 points)

A) Consider a contiguous memory system with memory allocated as shown below.



Suppose the following actions occur:

- Process E starts and requests 300 memory units. 1
- Process A requests 400 more memory units. 1 + 1/2
- Process B exits. 1/2
- Process F starts and requests 800 memory units. 1
- Process C exits. 1/2
- Process G starts and requests 900 memory units. 1 + 1/2

12

(a) Describe the contents of memory after each action using the first-fit algorithm. 6

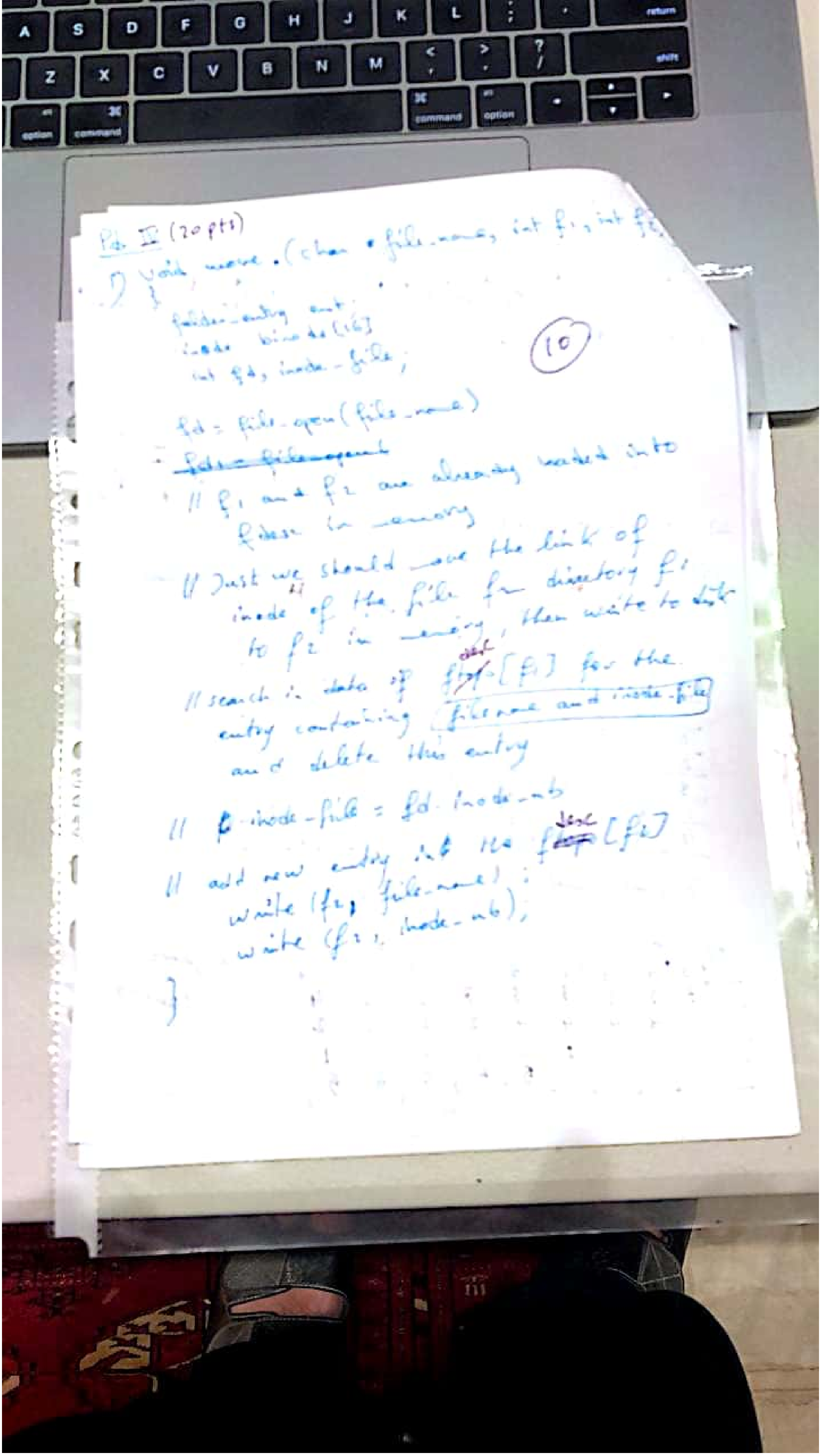
(b) Describe the contents of memory after each action using the best-fit algorithm. 5

(c) For this example, which algorithm is best? 1

P.S: you can compact the memory in case of need such that moving used blocks or free blocks

B) Consider a memory paginated system with page size 256 bytes. In this system each process is authorized max 4 frames in main memory. The page table of a process P1 is given in the following table:

page	0	1	2	3	4	5	6	7
frame	011	001	000	010	100	111	101	110
presence	1	0	1	0	0	0	1	0



Part II (20 pts)

```
1) void move (char * file_name, int f1, int f2
   {
      folder_entry ent;
      inode  binode[16]
      int fd, inode_file;

      fd = file_open(file_name)
      ~~fd = file_open~~
      // f1 and f2 are already loaded into
         fdesc in memory
      // Just we should move the link of
         inode of the file from directory f1
         to f2 in memory, then write to disk
      // search in data of ~~flag~~ dat[f1] for the
         entry containing filename and inode_file
         and delete this entry

      // f.inode_file = fd.inode_nb
      // add new entry at the ~~flag~~ desc[f2]
         write (f2, file_name);
         write (f2, inode_nb);
   }
```

10

Scanned by CamScanner

1) What is the size of the virtual address space of the process P1 2
2) What is the size of the physical memory? 2
3) Convert the following virtual addresses to physical (signal the error if any): 546, 2072. 2
4) What will happen if P1 generates the virtual address 770? 1

C) Consider a system with memory capacity 2GB, page size 4KB and addressing on 32 bits. Given that each table entry contains a reference to a frame + 1 bit presence/absence.

5
- a. What is the size of the page table (justify your answer)
- b. How many pages are needed to load the page table in memory?

D) Consider a process with virtual address space of 600 Bytes. the set of virtual addresses referenced is: 34; 123, 145; 510; 456; 345; 412; 10; 14, 12, 234. 336. 412

6
- a. Give the list of referenced pages given the size of the page is 100 Bytes
- b. Determine the number of page faults for the LRU algorithm. The memory is initially empty and contains 3 frames

Part 4: File System (40 minutes – 25 points)

A) Consider a file currently consisting of 100 blocks of data. Assume that the file control block is loaded in memory and there is no cache disk. The size of the block is 4KB. Calculate the number of disk I/O operations required for **contiguous** and **linked** allocation strategies to make the following changes to the file. In the contiguous case, you may assume there is no space to grow in the end. Also assume that the new information to be added to the file is not stored in memory.

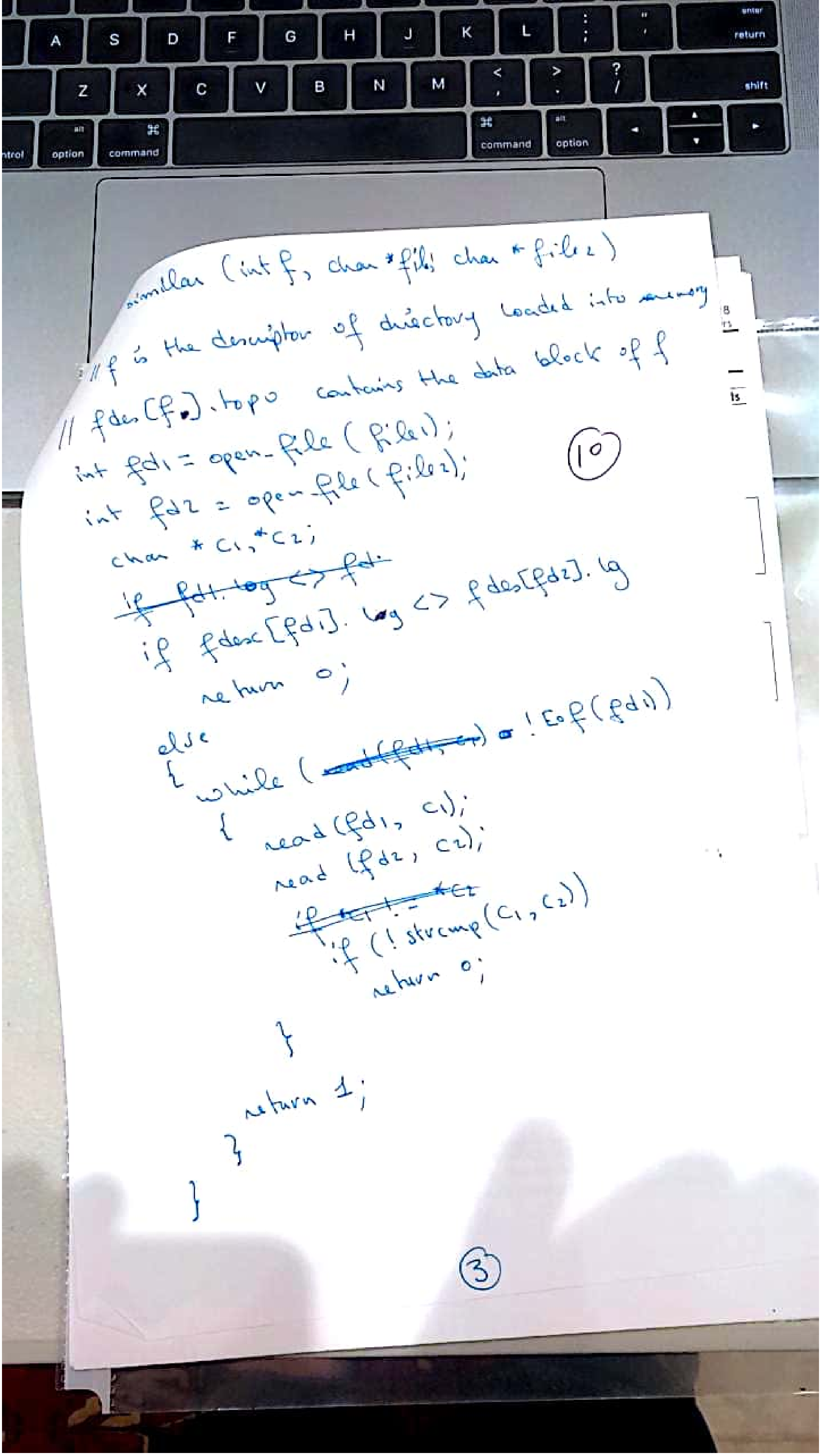
12
- a. **Add 2 blocks at the beginning**
- b. **Add 2 blocks at the end**
- c. **Remove the middle block**

B) Refer to the functions written in class (i.e., **create_inode**, ...):

6
- a. Describe (without writing code) the steps needed to create an inode 3
- b. How many **I/O disk** request is required to perform this task 3

C) Consider a disk of size 20GB in which the system installed is 16-bit DOS (FAT). The disk is divided into a set of blocks of fixed size (128KB). This disk contains 520 files: 200 files of size 16K, 200 files of size 256KB and 120 files of size 1Mb

7
- a. Calculate in MB the disk space. 1
- b. Calculate the number of blocks on disk 1
- c. How many blocks do occupy each of these three categories of files? 2
- d. Calculate in Kb the size of the FAT table. Justify your answer 3



```
similar (int f, char *file1 char * file2)
// f is the descriptor of directory loaded into memory
// fdes[f].topo   contains the data block of f
int fd1 = open_file (file1);
 int fd2 = open_file (file2);
  char * c1, *c2;
  ~~if fd1.log <> fd~~
   if fdesc[fd1]. log <> fdes[fd2]. lg
        return 0;
   else
   {  while ( ~~read(fd1, c1)~~ or ! Eof(fd1))
      {
          read (fd1, c1);
          read (fd2, c2);
           ~~if *c1 != *c2~~
           if (! strcmp(c1, c2))
               return 0;

       }
     return 1;
    }
}
```

10

3

INFO 3211 2014 - 2015

Part I

1) a) advantage of contiguous file allocation vs linked allocation policy

- quick and easy calculation of block holding data
  - just offset from start of file
- for sequential access, no seeks required
- the read performance is excellent bcz the entire file can be read from the disk in a single operation only one seek is needed (the first block).
- No problem of reliability whereas this is a big problem is linked allocation 2
- the amount of storage is a power of 2

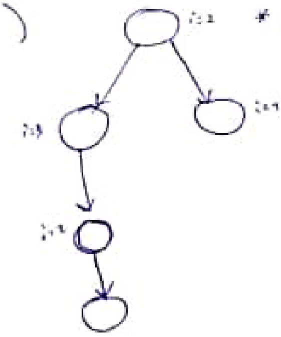
b) Problem of fragmentation

- During its lifespan, a process can request and free many chunks of memory.
- When a process is started, the free memory areas are long and contiguous.
- Over time and with use, the long contiguous regions become fragmented into smaller and smaller contiguous areas. Eventually, it may become impossible for the program to obtain large contiguous chunks of memory. 4
- there exist two types of fragmentation:

- internal: Due to the rules governing memory allocation (such as paging), more computer memory is sometimes allocated ~~external~~ than is needed. For example, in a system with page size 512Kbytes, a file with size 400 KB is allocated one page, and thus there is a loss of 112 KB => this waste is called internal frag- 1

Part I (continued)

B)



* there is 5 processes including the main process

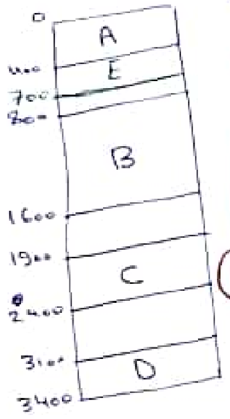
5

C) just one process is generated in this code by the parent (main).

the main process change the address space and executes another process and the child exits. 4

Part II (Memory management)

A)



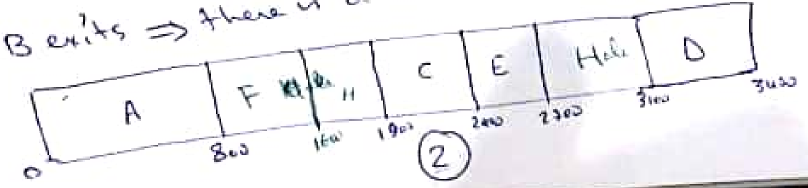
a) First Fit algorithm:

1 * process E starts and requests 300 memory units
=> allocation in the first free zone 400-700

* process A requests 400 more memory units ~~to process~~ A
=> ~~no free spaces to give to process~~
~~=> compact the memory~~
=> cannot fit because the entire process is allocated in a single continous chunck of memory in a contiguous memory system
=> Move B to 1100-1900
Move E to 2400-2700
* Give A additional addresses 400-800 between 800-1900 12

1

* B exits => there is a hole



2

→ <u>External</u> : arises when free memory is separated into small blocks and is interspersed by allocated —eory

- The term external refers to the fact that the unusable storage is outside the allocated regions
- For example in a contiguous allocation strategy with dynamic partitions, consider a situation where in ~~a programs~~ the memory allocator allocates 3 contiguous ~~blocks~~ region of memory for 3 different processes, and then free the middle region due to a swap or exit.

  if a new demanded region of —eory is larger than the free space ⇒ this free space is called external fragmentation

→ Internal fragmentation : view by the process

→ external frag—ti— : view by the syste—

<u>Remediation</u>

- there is no complete solution for internal frag—entations
- the internal frag—entation is always, at max with the size of a —eory page
- the external frag—entation can be avoided by ~~using~~ decomposing the memory into fixed size blocks such as <u>paging system</u>.

- process F starts and requests 800 memory units
  ⇒ F is allocated in 800 - 1600 (1)
  ⇒ there is a hole between 1600 - 2400 ($\frac{1}{2}$)
- C exists ⇒ there is a hole between 1600 - 2400
- G starts and requests 900 memory units
  ⇒ no hole that is big enough
  ⇒ compact memory : move E to 2800 - 3100 (1) + ($\frac{1}{2}$)
  ⇒ G is allocated in $\boxed{1600 - 2500}$

b) Best - Fit algorithm

E requests 300 ⇒ E is allocated in 1600 - 1900 (1)

A requests 400 more ⇒ this 400 is allocated in 400 - 800 (1)

B exits ⇒ there is a hole between 800 - 1600 (1/2)

F requests 800 ⇒ F is allocated in 800 - 1600 (1)

C exits ⇒ there is a hole between 1900 - 3100 (1/2)

G requests 900 ⇒ G is allocated in 1900 - 2800 (1)

c) Worst - Fit Algorithm (not required)

E requests 300 ⇒ E is allocated in 2400 - 2700

A requests 400 more ⇒ this additional 400 is allocated in 400 - 800

B exits ⇒ there is a hole between 800 - 1900

F requests 800 ⇒ F is allocated in 800 - 1600

C exits ⇒ there is a hole between 1600 - 2400

G requests 900 : no hole big enough ⇒ need to compact
⇒ move E to 2800 - 3100, give 1600 - 2500 to G.

For this exaple, $\boxed{\text{Best - Fit is the best}}$ (1)

C) Memory size = 2 GB

page size = 4 KB

addressing 32 bits

each PTE : reference to frame + 1 bit P/A

(a) <u>size of page table</u>

- addressing on 32 bits ⇒ 2^{32} virtual address space
- page size = 4 KB = 2^{12}

$$\Rightarrow \underbrace{}_{\text{page \#}}^{20}\ \underbrace{}_{\text{offset}}^{12}$$

⇒ page table contains 2^{20} entries

Or the number of frames is : $\frac{\text{Memory size}}{\text{Page size}} = \frac{2\,GB}{4\,KB} = \frac{2^{31}}{2^{12}}$

(3) $= 2^{19}$

⇒ we need 19 bits for the frame #

⇒ each PTE size is 19 + 1 = 20 bits

⇒ the size of page tables is $2^{20} \times 20$

= ~~$2^{22} \times 5$~~ 20MB

= ~~2559~~ KB

(b) <u>How many pages ?!</u>

$\#\text{pages} = \frac{\text{~~2559~~ KB} \ 20MB}{4\ KB} = \boxed{\cancel{6\,39}}$ ~~40~~ (2) pages 5120

D) process : virtual address space 600 Bytes page size = 100 Bytes

a) 0, 1, 1, 5, 4, 3, 4, 0, 0, 0, 2, 3, 4 (2)

9 page faults

Frame \ Pg			•				*		v	v			
f0	0	0	0	0	4	4	4	4	4	4	4	3	3
f1		1	1	1	1	3	3	3	3	3	2	2	2
f2				5	5	5	5	0	0	0	0	0	4

(4)

B) - page size = 256 Bytes

- each process is authorized max 4 frames

Page	0	1	2	3	4	5	6	7
frame	011	001	000	010	100	111	101	110
presence	1	0	1	0	0	0	1	0

1) Size of virtual address space of P1

$8 \times 256 = 2048$ bytes (1)

2) size of physical memory

the frames are coded in 3 bits ⇒ $2^3 = 8$ frames

⇒ physical memory size = nb(frames) × size(frame)

(2) $= 8 \times 256 = 2048$ bytes

$= 2KB$

3) convert to physical address

the conversion from virtual address to physical is realized as the followings:

a) calculation of page # and offset

b) search in page table the frame #

c) physical address frame address + offset

then :

$546 = 2 \times 256 + 34$

⇒ page 2 & offset 34

⇒ frame 0 (page table)

⇒ physical address is $\boxed{34}$ (1)

2072 is out of the virtual address space of P1

⇒ $\boxed{\text{error}}$ (1)

4) $770 = 3 \times 256 + 2$ ⇒ page 3. But this page is not loaded in memory ⇒ page fault (1)

(3)

Part 4 : File System

- file : 100 blocks of data
- File control block is loaded in memory
- size of block = 4KB

Number of disk I/O?

	contiguous	linked
Add 2 blocks at begin	2R + 2W ②	2R + 2W ②
Add 2 blocks at end	102 R + 102 W ②	3R + 3W ②
Remove the middle block	50R + 50W ②	~~23~~ 50 R + 1W ②

i) add 2 blocks at the begining

- contiguous : 2 reads block for new information
  + 2 write blocks "
- linked : - 2 reads for new inf
  - update pointers in memory
  - 2 writes for new inf

ii) add 2 blocks at the end

- contiguous : there is no room to add at the end
  => shifting all the blocks two places at the end
  => 100 R + 100 W
  - read 2 new inf blocks
  - write 2 new inf blocks } 2R + 2W
  $\Rightarrow \boxed{102 R + 102 W}$

- linked : - Read in the last block → 1R
  - Read in the 2 new blocks → 2R
  - update pointers in memory → 0R, 0R
  - write the three blocks → 3W
  } 3R + 3W

iii) Remove the middle block

C

- contiguous:

to delete position 50, read all blocks ~~before~~ after # 50 and write them back one place ~~closer to the front~~

⇒ 50 R + 50 W

- ~~read 2 new inf blocks and write it back~~

⇒ ~~2R + 2W~~

⇒ $\boxed{50R + 50W}$

- linked:

read an follow links to position 50

⇒ ~~0~~ 50 R → update in memory block 49 (Next) → 51

then write block 49 to link to former block 51 ⇒ 1 W

⇒ $\boxed{\overset{50}{\cancel{1,5}} R + 1 W}$

B) create_inode

10

a) the steps to create inode are:

- search for free inode:
  1- in cache
  2- not found, in disk

~~5~~ (3)

b

- bring the block that contains the inode to memory
- initialize the inode
- write back the block containing the inode to disk.

b) # Of disk I/O?

- if there is free inode in cache ⇒ ~~[illegible]~~.

  1R + 1W

- if no free inode in cache ⇒ search on disk

⇒ worst case read all blocks of inodes

⇒ $\boxed{isize\ R + 1 W}$

~~4~~ (3)