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?

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

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;}
```

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

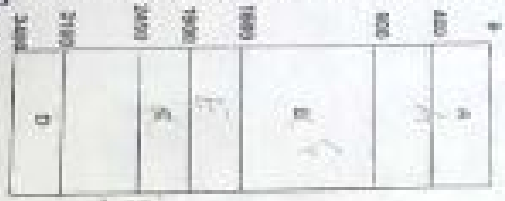
C) How many processes are generated by the following code :

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

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.
- Process A requests 400 more memory units.
- Process B exits.
- Process F starts and requests 800 memory units.
- Process C exits.
- Process G starts and requests 900 memory units.

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

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

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

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

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

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

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

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)

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.

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

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

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

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.

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

INFO 324

Part I

A) 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 beg 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 in linked allocation
- the amount of storage is a power of 2

(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 ~~than~~ than is needed. For example, in a system with page size 512 Bytes, a file with size 100 KB is allocated one page, and thus there is a loss of 12 KB => this waste is Internal frag-

(1)

- External : arises when free memory is separated into small blocks and is interspersed by allocated —
- 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 program~~ the memory allocator allocates 3 contiguous ~~blocks~~ regions of memory for 3 different processes, and then free the middle region due to a swap or exit.

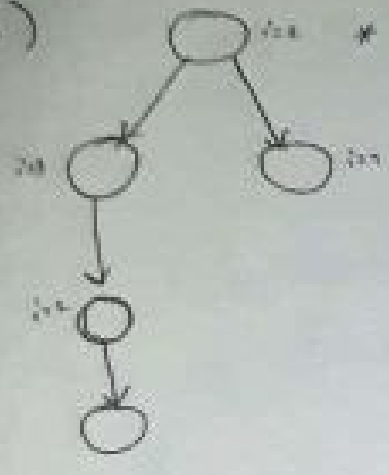
if a new demanded region of — ory is larger than the free space ⇒ this free space is called external fragmentation

→ Internal fragmentation : view by the proces
→ external frag — : view by the syste

~~The~~ Remediation

- there is no complete solution for internal fragmentation
- the internal fragmentation is always, at max with the size of a — ory page
- the external fragmentation can be avoided by ~~using~~ decomposing the memory into fixed size blocks such as paging system.

I (continued)

B) 

* There is 5 processes including the main process

(5)

C) Just one process is generated in this code by the fork() (main).

the main process changes the address space and executes another process and the child exits. (4)

Part II (Memory management)

a) First Fit algorithm:

A) 

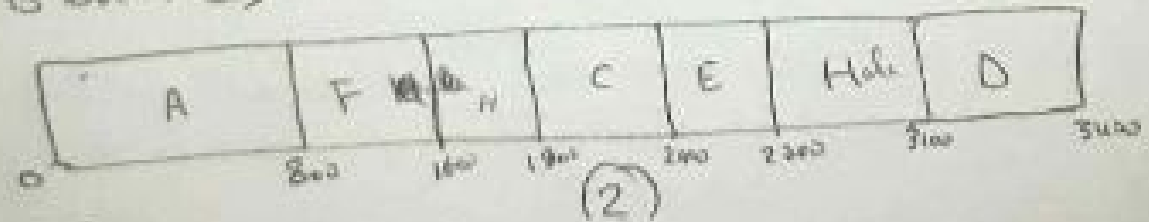
① * process E starts and requests 300 memory units
⇒ allocation in the first free zone 1100-1400

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

(1)

(1½)

* B exits ⇒ there is a hole between 800-1900



(2)

* process F starts and requests 800 memory units
⇒ F is allocated in 800 - 1600 (1)
* C exits ⇒ there is a hole between 1600 - 2400 (1/2)
* G starts and requests 300 memory units
⇒ no hole that is big enough
⇒ Compact memory: move E to 2800 - 3100 (1) + (1/2)
⇒ G is allocated in 1600 - 2500

b) Best-Fit algorithm

E requests 300 ⇒ E is allocated in 1600 - 1900 (1)
A requests 100 more ⇒ this 100 is allocated in 100 - 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 100 more ⇒ this additional 100 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 example, Best-Fit is the best (1)

- 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 $\Rightarrow 2^3 = 8$ frames

$\Rightarrow$ physical memory size = nb(frames) × size(frame) (2)

$= 8 \times 256 = 2048$ bytes

$= 2$KB

3) convert to physical address

The conversion from virtual address to physical is realized as the following:

a) calculation of page # and offset

b) search in page table the frame #

c) physical address = frame address + offset

Here:

$546 = 2 * 256 + 34$

$\Rightarrow$ page 2 & offset 34

$\Rightarrow$ frame 0 (page table)

$\Rightarrow$ physical address is 34 (1)

2072 is out of the virtual address space of P1

$\Rightarrow$ error (1)

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

(2)

C) Memory size = 2 GB
page size = 4 KB
addressing 32 bits
each PTE is reference to frame + 1 bit p/A

(a) size of page table

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

⇒ 20 (page #) | 12 (offset)

⇒ page table contains 2^{20} entries

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

(b) $= 2^{19}$

⇒ we need 19 bits for the frame #

⇒ each PTE size is 19 + 1 = 20 bits

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

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

$= 2559$ KB

(b) How many pages ? 20 MB

pages = $\frac{2559\ KB}{4\ KB} = 639$ pages (2)

5120

D) process's 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 \ Ref			*				*		*	*			
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)

FS = 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 beginning	2R + 2W (1)	2R + 2W (2)
add 2 blocks at end	102R + 102W (3)	3R + 3W (4)
remove the middle block	50R + 50W (5)	~~50~~ R + 1W (6)

i) add 2 blocks at the beginning

- 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
  => 100R + 100W
  - read 2 new inf blocks
  - write 2 new inf blocks } 2R + 2W
  => 102R + 102W

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

A) Remove the middle block

- contiguous:

to delete portion 50, need all blocks after ~~it~~ ⇒ 50 and write them back one place ~~closer to the front~~

⇒ 50 R + 50 W

- ~~need 2 more inf block and write it back~~

⇒ ~~the 2nd~~

⇒ 50R + 50W

- linked:

need an follow links to portion 50

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

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

⇒ 50R + 1 W

B) create - inode

a) the steps to create inode are:

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

(3)

- bring the block that contains the inode to ~~a~~ any
- initialize the inode
- write back the block containing the inode to disk.

b) # of disk I/O?

* if there is free inode in cache ⇒
* 1R + 1W
* if no free inode in cache ⇒ search on disk
  ⇒ worst case read all blocks of inode
  ⇒ size R + 1 W

(3)

disk: - size = 10 GB
- 16 bit DOS
- block = 128 KB
- 520 Files → 200 files (16K)
→ 200 files (256 KB)
→ 120 files (1 MB)

a) disk space = 20×2^{10} MB = 20480 MB (1)

b) Number of blocks = $\frac{\text{disk size}}{\text{block size}} = \frac{20 \times 2^{10} \times 2^{10}}{128}$ KB

= 163840 blocks

$= 20 \times 2^{13}$ (1)

$= 5 \times 2^{15}$

c) - 200×16 KB = 200×2^4 KB

= 3200 KB

⇒ 25×128 KB = 25 blocks (not real solution but I'll accept)

other: in fact each file needs one block so 200 blocks; only I can't consider that 2 solutions I have

- 200×256 KB ⇒ 400 blocks (1)

- 120×1 MB ⇒ 120×8 = 960 block

d) Size of the FAT table ?

Size (FAT) = Number of entries * size (entry)

or FAT on 16 bits ⇒ max 2^{16} clusters (3)

= 65535 clusters

or each cluster must contain power of 2 sectors

⇒ each cluster must contains 4 blocks

⇒ the entry in FAT is 18 bits

$\Rightarrow$ size (FAT) $= 2^{16} \times 18 =$ **143 KB**

[illegible] University of Sciences	INFO 324 Operating System II	9 June 2016 Duration : 2 hours

Answer the following questions with justification :

1. Explain the difference between segmentation and pagination, the relative interest of each other
2. What is virtual memory? give the reasons why it is advantageous to have a virtual memory mechanism on a computer
3. What is a fragmentation and how to remedy it in contiguous memory allocation

Consider the following C program:

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

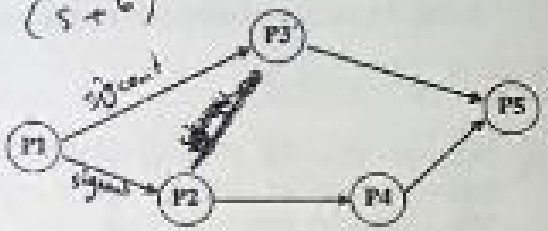
1. The graph generated by calling f(3).
2. Draw the graph generated by the following program

```
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 created in part 2.

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.

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



Exercise II

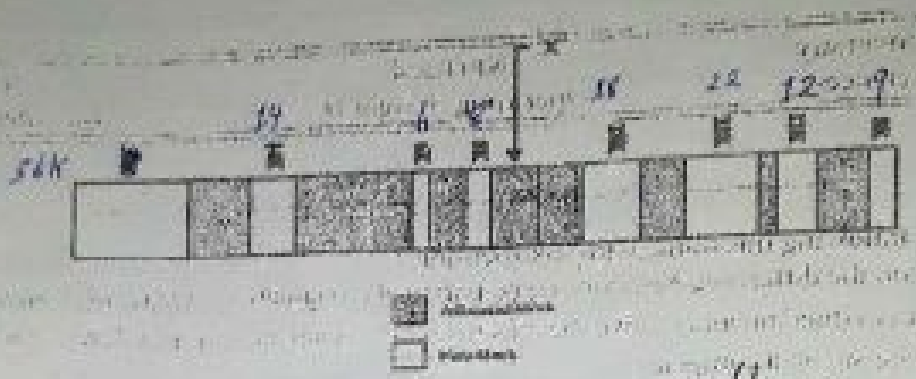
Consider a paginated memory with size of 48 KB and pages of size 12 KB. The following references are requested in memory: A, B, C, D, E, B, E, F, D, A, B, C, G, F, C, B, A, B, C, F
How many page faults are generated using the following replacement algorithms?

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

Exercise III

We wish to allocate memory space for a process of 16KB.

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 PI (6 pts)

Segment	Base	Limit
0	540	234
1	1254	128
2	54	328
3	2048	1024
4	926	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

(4) 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.)

(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.

A) 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

5 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.

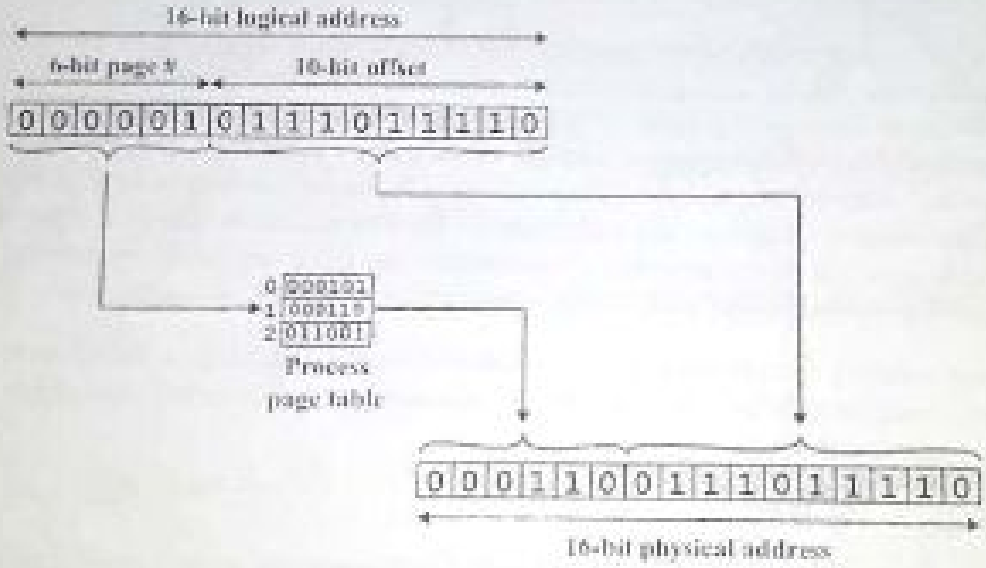
- Paging does not distinguish and protect procedures and data separately.
- Segmentation distinguishes and separately protects procedures and data.
- Unlike segmentation, Paging does not facilitate sharing of procedures.
- Paging is transparent to programmers (system handles it automatically).
- Segmentation 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:

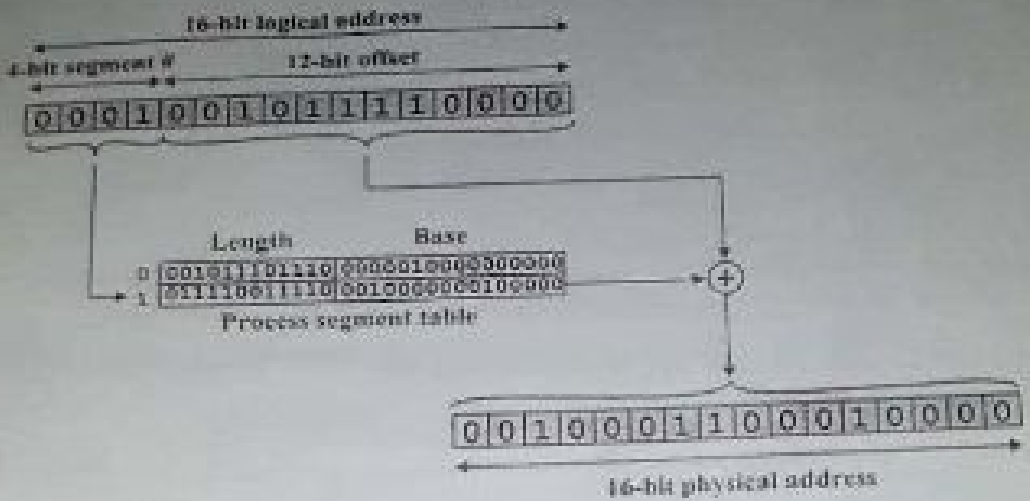
For paging:

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) **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

S.N.	Fragmentation	Description
1	External fragmentation	Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous so it cannot be used.
2	Internal fragmentation	Memory block assigned to process is bigger. Some portion of memory is left unused as it cannot be used by another process.

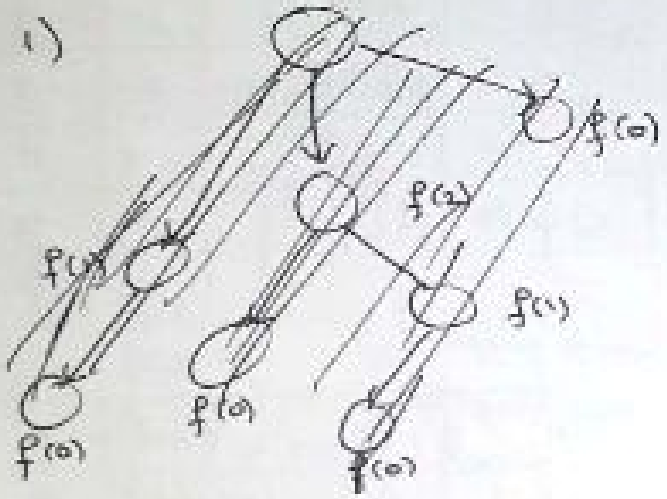
External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

Exercice I (27 pts : A → 8 (3+2+3)
B → 8 (3+5+1)
C → 11 (5+6)

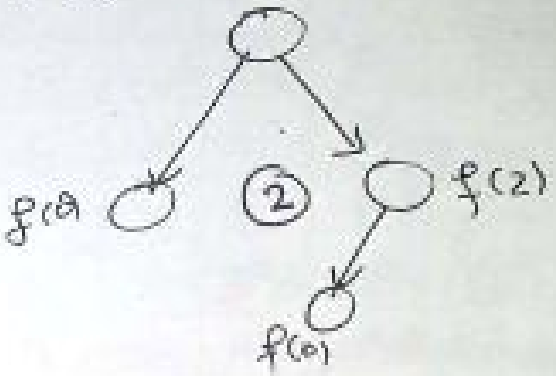
B) void f(int i)
{
if (i<0) exit(0);
if (i%2 == 0) fork();
f(i-1);
}

f(3) → f(2) —fork()→ f(1) → f(0) —fork()→ f(-1) ×

1)



4 processus



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

f(0) —fork()→ f(-1) ×
f(1) → f(0)
f(2)
f(3)

⇒

i=0 f(0) —fork()→ f(-1) ×
i=1 f(1) → f(0) —fork()→ f(-1) ×
i=2 f(2) —fork()→ f(1) → f(0) —fork()→ f(-1) ×
i=3 f(3) → f(2) —fork()→ f(1) → f(0) —fork()→ f(-1) ×

we have 6 calls to fork, so 2^6 processes = 64 (5)

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

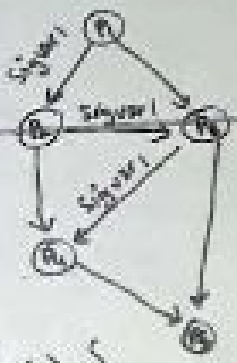
1

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

c) #include <stdio.h>
1) #include <stdlib.h>
#include <unistd.h>
void main() {
int P12[2], P23[2], P31[2],
P45[2];
int x;
pipe(P12); Pipe(P23); ----

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



6 pts

void main() {
pipe(fd);
signal(sigusr1, handler);
for (int i=1; i<=4; i++)
{
if(!(pid = fork()))
{ close(fd[1]);
pause();
}
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++;

11

Memory :- size 48 KB
- page 12 KB
} ⇒ 4 Frames

a) LRU (5) (1+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	F	D	
C	yes	B	A	C	D	
G	yes	B	A	C	G	
F	yes	B	F	C	G	
C	NO	B	F	C	G	
B	NO	B	F	C	G	
A	yes	B	F	C	A	
B	NO	B	F	C	A	
C	NO	B	F	C	A	
E	NO	B	F	C	A	

12 page faults

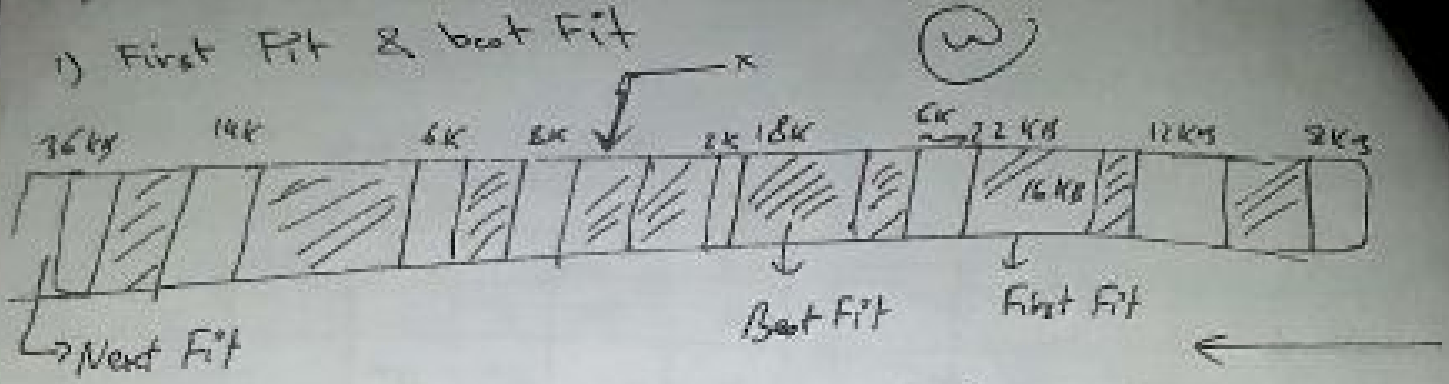
b) Second chance (5) (1+2)

Page ref	4 Page Frames				
	Faults	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*	E	F	B
A	yes	A*	D	E*	F*
B	yes	B*	A	E	F
C	yes	C*	B	A	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	B
B	NO	A	F	C	B*
C	NO	A	F	C*	B*
F	NO	F*	C*	B*	A*

10 page Faults
12 → see next Page

B) process = 16KB

1) First Fit & best Fit



C) 1) (0; 128) → 540 + 128 = 668

① (1 : 100) → 1254 + 100 = 1354

(2 : 465) → error

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, 8100, 1000, 1200

a) FCFS → 36400

b) SSTF → 9400

c) Scan → 9600 or 18498

d) C-look → 13400 or 16000

5360

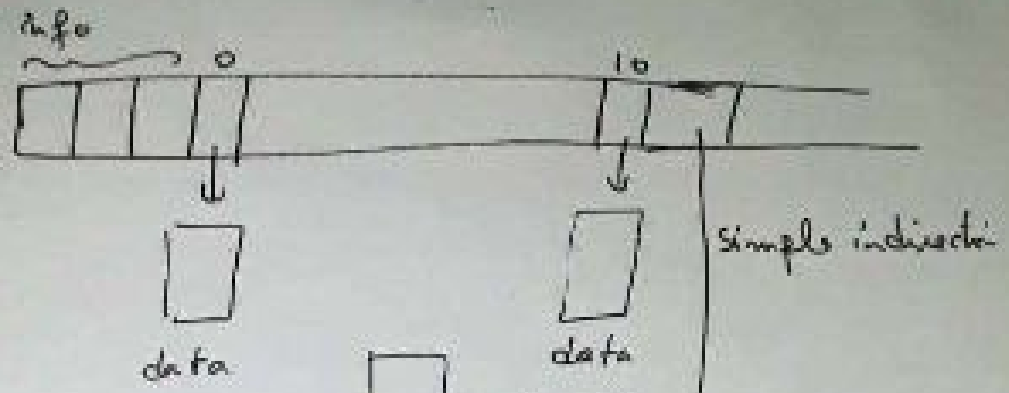
5360

x 9600

Second Chance Alg

Page	Fault	4 pages 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



Inode :

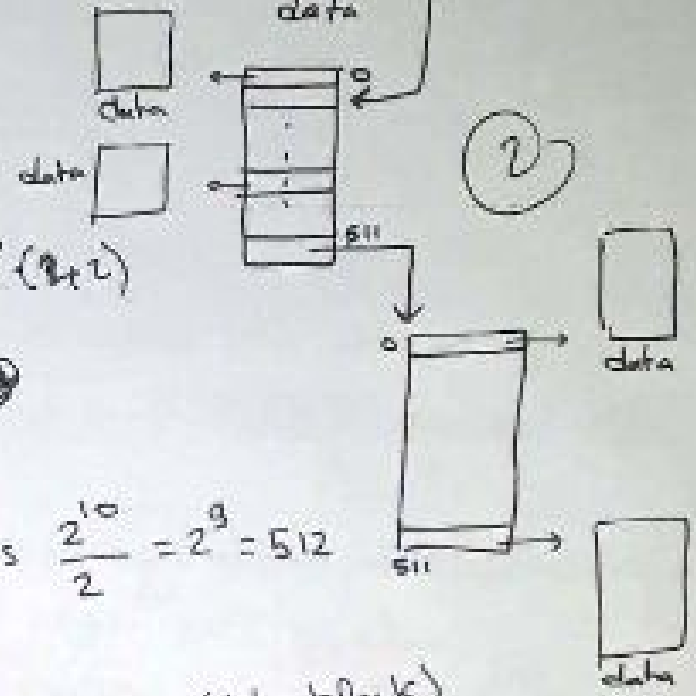
(7) (11+1)

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 + 512 (data block) + 512 (data block)

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

11264

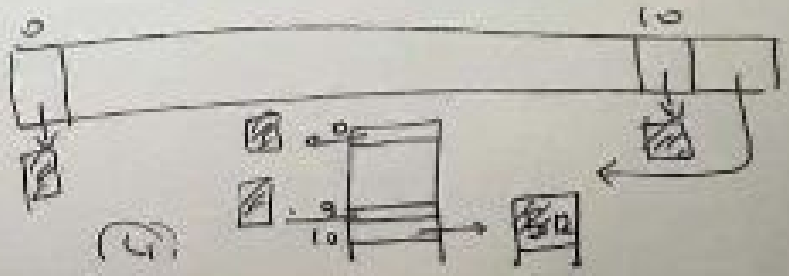
b) $20992 - (11 \times 1024) = 20992 - 11264 = 9728$

$9728 = (9 \times 1024) + 512$

9216

So the file covers the 11 direct data block - the first 0 ⟶ 9 (10) from the simple indirecta - In the block 10 (simple indirection), we have

offset 512 (3)



(4)

c) file_open (10 = 9 + 1)

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 (name_file, inode)

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

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

(6)

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

6- initialize the fdesc entries

~~inode~~

~~eof~~

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 [x] I/O

(x) * disk_read (binode, 2 + (fold.inode_nb/16)); → 2 I/O

So we need : [x + 2 I/O]