LEBANESE UNIVERSITY
Faculty of Sciences – Section I
Info 3303-324 - E

Final 2017-2018
Duration : 2 hours

Part I — Process Management — 20 Points

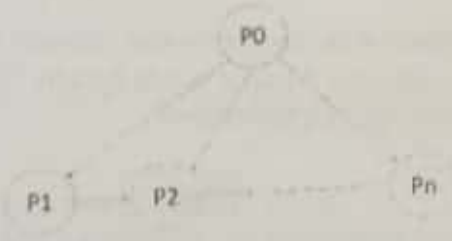
The goal of this problem is, starting from a **sorted set of integers**, filter it successively to keep only prime numbers. Consider integers from the number 2 (which is the first prime number). To build the rest of the prime numbers, let's first remove the multiples of 2 from the rest of the integers, we get a list of integers starting with 3 which is the next prime number. Let's eliminate the multiples of 3 from this list, which builds a list of integers starting with 5, and so on.

Write a C under UNIX program to make this filter where each filter is a process that reads a list of integers, displays the first prime number p on screen, and leaves the integers it has not filtered. The main process write the list of integers into a pipe.
Two solutions are requested as follows:

a) **Sequential processing**: at the beginning, only the main process is created, it creates the first child that displays the first prime number (that is 2) and filters its multiples, then the child process gives the hand to another child process created by itself and so on.

P0 — P1 — P2 ..... Pn

b) Now, the main process creates a sufficient **number of processes** (to be determined) to display prime numbers between 2 and 120. These processes are in "pause" state after creation. The father process wakes up the first child process which displays the first integer first and eliminates its multiples. The first child wakes up the second child process to display the second prime number and filters its multiples and so on. The last process must display all prime numbers that do not have multiples in the list of integers between 2 and 120.



P.S: remember that reading from pipe is erasing

Part II — File Management — 25 Points

Given a FS (File system) where the table topo contains 10 entries, the first one points to a data block and the other nine entries have one level of indirection. Knowing that each block occupies 4 kilobytes, and the number of a block occupies 4 bytes and each block inode contains 16 inodes:

A)

a. Which is the maximum size of a file supported by this FS?

b. How much, the file of maximum size, does it occupy of *effective* space on the disc?
c. We consider a file containing 8,000,000 bytes. How much blocks (data and map) it is necessary to represent this file on disk? Justify briefly your answer.
d. Define the structure file descriptor (fdesc) correspondent to this FS.

B) Refer to the function **block_release()** written in class, then describe the steps to do for releasing a block without writing the code and calculate the number of I/O disk access (worst case) required to release a block.

Part III — Memory Management — 25 Points

1. Given free memory partitions of 300k, 125k and 260k (in this order), describe, step by step, how each of the algorithms First-fit, Next Fit and best-fit allocate processes of 100k, 250k and 250k (in this order).

2. Let **m** the number of memory cases allocated to a program and **W** the set of referenced pages in a given time. Let S (m, W) the set of pages present in memory after the set of references W. Consider a program with a size of 5 pages. These pages designated as A, B, C, D, E are initially in secondary memory (disk).

   a. Construct the set S (m, ABCDABEABCDE) with m = 4 in the case of a FIFO replacement algorithm. How many page faults resulting from this series of references?
   b. The same question of a) for m=3.
   c. The same question of a) and b) in the case of LRU page replacement algorithm.

3. Consider a memory system with pagination that contains 8 frames (frames) of 16 bytes (each), and a process occupying 4 pages in this system. The placement of pages in memory is represented in the diagram below.

Page 1
Page 0
Page 3
Page 2

   a. Indicate the contents of the page table of this process.
   b. Explain how the logical address 35 bytes (decimal) is converted into a physical address (specify the decimal value of this physical address).

Ex1 - a (10 points)

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

#define N 120

void main() {

int fd1[2],fd2[2],fd3[2],fd4[2],fd5[2],x,y,i,j;
int flag=0;
pipe(fd1);pipe(fd2);pipe(fd3);pipe(fd4);pipe(fd5);

for (i=2;i<=N;i++)
   write(fd1[1],&i,sizeof(int));

for(i=1;i<=5;i++)
{
  if(!fork()) {
    switch(i) {
       case 1:
          read(fd1[0],&x,sizeof(int));  //read 2
          printf("%d ",x);
          for(j=3;j<=N;j++) {
             read(fd1[0],&y,sizeof(int));
             if(y%x!=0) {write(fd2[1],&y,sizeof(int));}
          }//end for
       close(fd2[1]);close(fd1[0]);

       case 2:
          read(fd2[0],&x,sizeof(int)); //read 3
          printf("%d ",x);
          for(i=5;i<=N;i++) {
             if(read(fd2[0],&y,sizeof(int))>0){
                  if(y%x!=0) {write(fd3[1],&y,sizeof(int));}}
              else {break;}

          }//end for
     close(fd3[1]);close(fd2[0]);

       case 3:
          read(fd3[0],&x,sizeof(int)); // read 5
          printf("%d ",x);
          for(i=7;i<=N;i++) {
             if(read(fd3[0],&y,sizeof(int))>0){
                 if(y%x!=0) {write(fd4[1],&y,sizeof(int));}}
             else {break;}
          }//end for
      close(fd4[1]);close(fd3[0]);

      case 4:
          read(fd4[0],&x,sizeof(int));//read 7
          printf("%d ",x);
          for(i=11;i<=N;i++) {
             if(read(fd4[0],&y,sizeof(int))>0){
                  if(y%x!=0) {write(fd5[1],&y,sizeof(int));}}
             else {break;}

          }//end for
        close(fd5[1]);close(fd4[0]);
       case 5:
          close(fd2[0]);close(fd2[1]);close(fd3[0]);close(fd3[1]);close(fd5
[1]);close(fd4[1]);close(fd4[0]);close(fd1[1]);close(fd1[0]);
          for(i=11;i<=N;i++) {
```

```
          // printf("hdgsgdgsfgdfsfd\n");
           if(read(fd5[0],&y,sizeof(int))>0){
              printf("%d ",y);}
           else {break;}

        }//end for
      close(fd5[1]);close(fd4[0]);
   }// end switch
  }
  else {break;}
}//end main loop

}//end main function
```

Ex1-5 (10 pts)

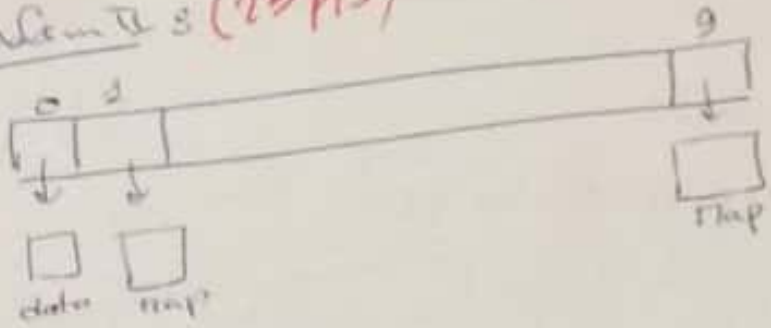
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#define N 120

static int cnt=0;
void handler(int nsig){cnt++; printf("Signal number %d arrived
\n",cnt);}
void main() {

int fd1[2],fd2[2],fd3[2],fd4[2],fd5[2],x,y,i,j,pid,p;
int flag=0; int fd[2];
pipe(fd1);pipe(fd2);pipe(fd3);pipe(fd4);pipe(fd5);pipe(fd);

for (i=2;i<=N;i++)
   write(fd1[1],&i,sizeof(int));

signal(SIGUSR1, handler);

for(i=1;i<=5;i++) {
if(!(pid=fork())) {pause();}
else {write(fd[1],&pid,sizeof(int));}
}

read(fd[0],&p,sizeof(int));
kill(p,SIGUSR1); //awake the first process
while(wait(NULL));

for(i=1;i<=5;i++)
{
  if(!pid) {
    switch(i) {
       case 1:
          read(fd1[0],&x,sizeof(int));  //read 2
          printf("%d ",x);
          for(j=3;j<=N;j++) {
             read(fd1[0],&y,sizeof(int));
             if(y%x!=0) {write(fd2[1],&y,sizeof(int));}
          }//end for
          close(fd2[1]);close(fd1[0]);
          read(fd[0],&p,sizeof(int));
          kill(p,SIGUSR1); //awake the second process
          exit(0);
       case 2:
          read(fd2[0],&x,sizeof(int)); //read 3
          printf("%d ",x);
          for(i=5;i<=N;i++) {
             if(read(fd2[0],&y,sizeof(int))>0){
                  if(y%x!=0) {write(fd3[1],&y,sizeof(int));}}
              else {break;}

          }//end for
        close(fd3[1]);close(fd2[0]);
          read(fd[0],&p,sizeof(int));
          kill(p,SIGUSR1); //awake the third process
```

INFO324 - 2017-2018
Solution

```
        exit(0);
      case 3:
        read(fd3[0],&x,sizeof(int)); // read 5
        printf("%d ",x);
        for(i=7;i<=N;i++) {
           if(read(fd3[0],&y,sizeof(int))>0){
              if(y%x!=0) {write(fd4[1],&y,sizeof(int));}}
           else {break;}
         }//end for
        close(fd4[1]);close(fd3[0]);
        read(fd[0],&p,sizeof(int));
        kill(p,SIGUSR1); //awake the fourth process
        exit(0);


      case 4:
          read(fd4[0],&x,sizeof(int));//read 7
          printf("%d ",x);
          for(i=11;i<=N;i++) {
             if(read(fd4[0],&y,sizeof(int))>0){
                if(y%x!=0) {write(fd5[1],&y,sizeof(int));}}
             else {break;}

          }//end for
          close(fd5[1]);close(fd4[0]);
          read(fd[0],&p,sizeof(int));
          kill(p,SIGUSR1); //awake the fifth process
          exit(0);

         case 5:
     //       close(fd2[0]);close(fd2[1]);close(fd3[0]);close
(fd3[1]);close(fd5[1]);close(fd4[1]);close(fd4[0]);close(fd1
[1]);close(fd1[0]);
            for(i=11;i<=N;i++) {
               if(read(fd5[0],&y,sizeof(int))>0){
                  printf("%d ",y);}
               else {break;}

            }//end for
          close(fd5[1]);close(fd4[0]);
          exit(0);
       }// end switch
    }
    else {break;}
  }//end main loop

  }//end main function
```

Solution

Problem II : (25 pts)



- 1 block = 4KB
- block # = 4 Bytes
- each block inode contains 16 inodes

A) a) max size of a file is : (5 pts)

each map can contain $\frac{4\,KB}{4\,B}$ = 1K entries = 1024 entries

$\Rightarrow$ max size = 4KB + (9 × 1024 × 4KB)

b) effective space on disk (5 pts)

(1 + 9 × 1024) → data
+
9 map block

} 10 + 9 × 1024 blocks

$\Rightarrow$ 10 + 9216 = 9217 blocks

$\Rightarrow$ 9217 × 4096 = 37752832 bytes

c) file → 8 000 000 bytes (5 pts)

8 000 000 = (1953 × 4096) + 512 bytes (1954 data blocks)

= 1 + (1024 × 4096) + (929 × 4096)

Nb Data blocks = 1 + 1024 + 929 = 1954 data blocks

Nb map block = 2 map blocks

B) block_release (7 pts)

- if table of free calls not full
- => add the block nb to this table

~~else~~

- increment the pointer
- if not : write the block to disk
  - set bfree = 0
  - set the next_table pointer to nb_block

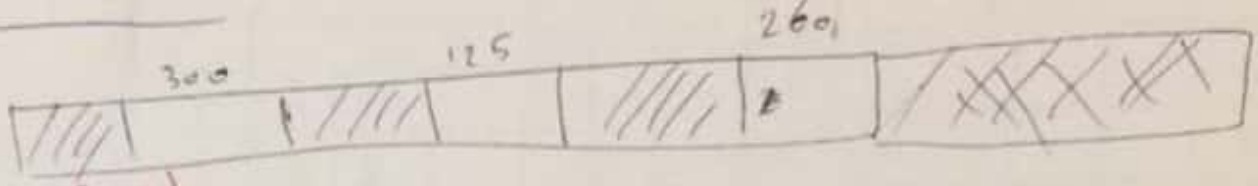
=> Just one disk access

~~Freeblock~~

~~(3 pts)~~

d)
```
struct fdesc {
        int topo[10];
        int map[1024];
        char buffer[4096];
        :
        }
```

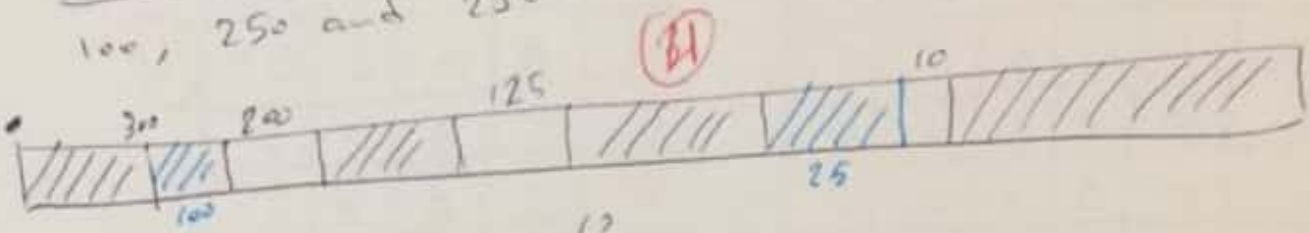
Problème II :



1) (6 pts)

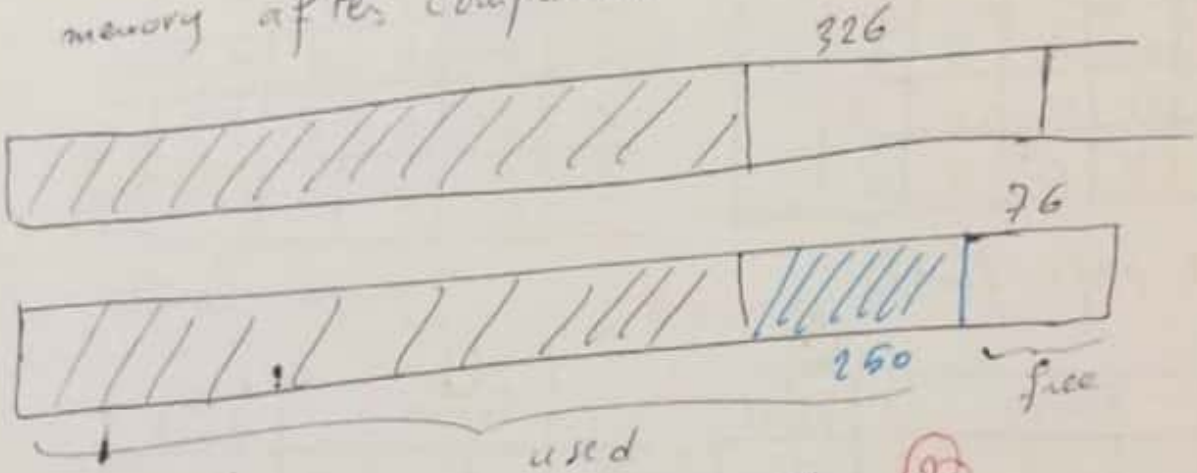
First Fit

100, 250 and 250



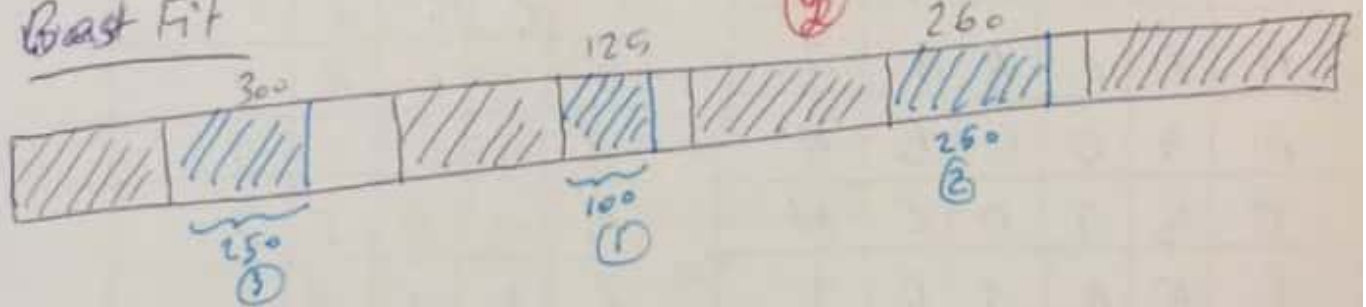
⇒ we need compaction

memory after compaction



Next Fit : same as First Fit (2)

Best Fit (2)



2) process → 5 pages

memory → m frames

m = 4 w = {A, B, C, D, A, B, E, A, B, C, D, E}

FIFO

	4 Frames				Page fault
A				A	y
B				A	y
C		C	B	A	y
D	D	C	B	A	y
A	D	C	B	A	N
B	D	C	B	A	N
E	E	D	C	B	y
A	A	E	D	C	y
B	B	A	E	D	y
C	C	B	A	E	y
D	D	C	B	A	y
E	E	D	C	B	y

(2)

10 page faults

	Frames			Page fault
D			A	y
B		B	A	y
C	C	B	A	y
D	D	C	B	y
A	A	D	C	y
B	B	A	D	y
E	E	B	A	y
A	E	B	A	N
B	E	B	A	N
C	C	E	B	y
D	D	C	E	y
E	D	C	E	N

(2)

9 page faults

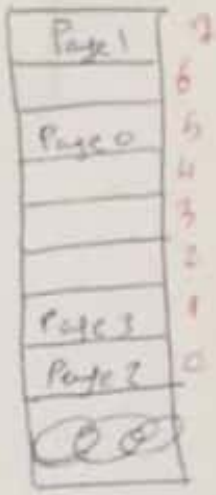
	(3) Frames			Page fault
A			A	y
B		B	A	y
C		C	B	A	y
D	D	C	B	A	y
A	A	D	C	B	N
B	B	A	D	C	N
E	E	B	A	D	y
A	A	E	B	D	N
B	B	A	E	D	N
C	C	B	A	E	y
D	D	C	B	A	y
E	E	D	C	B	y

8 page faults

	(3) Frames			Page fault
A			A	y
B		B	A	y
C	C	B	A	y
D	D	C	B	y
A	A	D	C	y
B	B	A	D	y
E	E	B	A	y
A	A	E	B	N
B	B	A	E	N
C	C	B	A	y
D	D	C	B	y
E	E	D	C	y

10 page faults

3)



Page table

Page	Frame	V
0	5	1
1	7	1
2	0	1
3	1	1

each Frame 16 Byte => 4 bits

P# offset
3bits 4bits
7 bits

b) convert 35 → physical adress

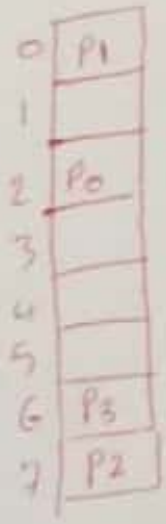
35 byte → page 2 ⇒ frame 0 ⇒ (0, 3)

(2, 3)

⇒ 3 byte (Physical)

4

05



Page	Frame	Presence
0	2	1
1	0	1
2	7	1
3	6	1
4	Null	0
5	Null	0
6	Null	0
7	Null	0

35 byte → page 2
offset 3

(2, 3) → (7, 3) Physical ⇒ adress = (7 × 16) + 3

logical

= 112 + 3

= (115) dec