1. Lets a system with 2Mbytes of main memory managed with partitions of varying size and that selects processes from the input with FIFO policy to locate them in memory. To initialize the system the OS (150 KB) is loaded in the lower memory locations. Then the execution of 5 processes start in the following order: P1(200KB), P2(450KB), P3(150KB), P4(250KB) and P5(650KB), memory requirements are indicated between brackets. After finishing P2 and P4, the processes P6(250KB), P7(150KB) and P8(300KB) arrive in that order.

a) Use a Best fit policy, and indicate the status of memory, occupation and gaps after allocating P8 in memory.

b) Use a Worst fit policy, and indicate the status of memory, occupation and gaps, after allocating P8 in memory.

(1.0 point)

1

a) Best fit

0 2048K

S.O. 150K	P1 200K	free 450K	P3 150K	free 250K	P5 650K	Free 198K

P6(250K)

0 2048K

S.O. 150K	P1 200K	free 450K	P3 150K	P6 (250K)	P5 650K	Free 198K

P7(150K)

0 2048K

S.O. 150K	P1 200K	free 450K	P3 150K	P6 250K	P5 650K	P7 150K	Free 48K

P8(300K)

0 2048K

S.O. 150K	P1 200K	P8 300K	free 150K	P3 150K	P6 250K	P5 650K	P7 150K	Free 48K

b) Worst fit

0 2048K

S.O. 150K	P1 200K	free 450K	P3 150K	free 250K	P5 650K	Free 198K

P6(250K)

0 2048K

S.O. 150K	P1 200K	P6 250K	free 200K	P3 150K	free 250K	P5 650K	Free 198K

P7(150K)

0 2048K

S.O. 150K	P1 200K	P6 250K	free 200K	P3 150K	P7 150K	free 100K	P5 650K	Free 198K

P8(300K)

P8 process could not be allocated in memory, because there is not a gap (contiguous space) greater or equal to 300 K. Compaction could be used since the sum of the sizes of the free holes 200 K + 100 K + 198 K is enough to allocate P8

2. A system has a 2 GB logical address space, 2 KB pages and 1 GB of physical memory.

a) Calculate the physical address corresponding to the logical address 4119, considering that memory is managed by paging. The first entries in the page table are shown below. Explain your answer.

Page table

Page	Frame	Valid bit
0	27	v
1	8	v
2	500	v
3	0	v

b) Calculate the logical address generated by a process that corresponds to the physical address 10288, considering segmentation with paging with a maximum segments number of 64 segments per process. The page tables for segments 0, 1 and 2, of this process are:

Page table segment 0

Page	Frame	Valid bit
0	2	v
1	800	v
2	1024	v
3	3	v

Page table segment 1

Page	Frame	Valid bit
0	8	v
1	0	v
2	328	v
3	5	v

Page table segment 2

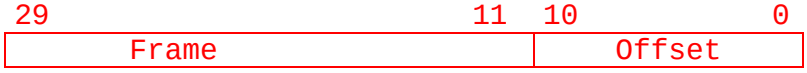
Página	Marco	Valid bit
0	500	v
1	21	v
2	1	v
3	82	v

(1.25 points)

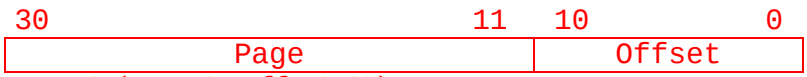
2

- Physical space de 1GB --> 30 bit physical address
- Page size = Frame size = 2KB --> 11 bit offset
- Logical space 2GB --> 13 bit logical address
- Number of bits for p2 is 31-11 = 20 bits

Physical address



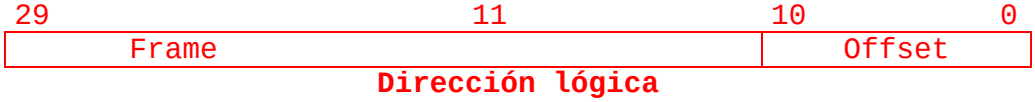
Dirección lógica



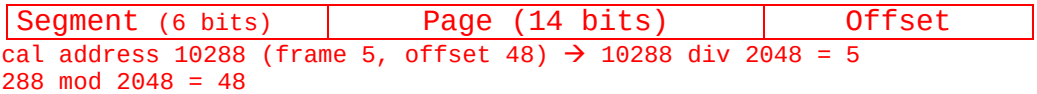
Logical address 4119 (page 2, offset 23)
4119 div 2048 = 2 → belongs to page 2
4119 mod 2048 = 23 → the offset is 23
Checking the page table → page 2 descriptor contains frame 500
Physical address: 500 * 2048 + 23 = **1.024.023**

- Physical space 1GB --> 30 bit physical address
- Page size = Frame size = 2KB --> 11 bit offset
- Logical space 2GB --> 31 bit logical address
- Maximum nº of segements 64 --> 6 bit segment id
- Number of bit for page field 31-11-6 = 14 bits

Physical address



Dirección lógica



Physical address 10288 (frame 5, offset 48) → 10288 div 2048 = 5
10288 mod 2048 = 48
Checking the segment tables → frame 5 appears in segment 1 table for page 3, so the logical address that has generated the physical address is:
(1, 3*2048+ 48) = (1, 6192)

3. Let an operating system that manages virtual memory using paging with 4 KB pages. Frames free, if any, are assigned in increasing order of physical addresses. At a given time, the system has distributed 3 frames (0, 1 and 2), that are initially free, between two new processes A and B.

a) Applying an LRU replacement algorithm with global replacement, indicate the evolution of physical memory contents if the following pages are referenced: A0, B0, A1, A0, A2, A1, B0, A3, B3. Obtain the number of page faults that occur.

b) By applying a second chance replacement algorithm with global replacement, indicate the evolution of the contents of physical memory if the following pages are referenced: A0, B0, A1, A0, A2, A1, B0, A3, B3. Indicate the number of page faults that occur.

c) Explain if the LRU page replacement algorithm suffers of Belady's anomaly.

Explain all your answers.

(2,0 points)

3

a) a) LRU with global replacement

	A0	B0	A1	A0	A2	A1	B0	A3	B3
marco	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
0	A0 (0)	A0(0)	A0(0)	A0(3)	A0(3)	A0(3)	B0(6)	B0(6)	B0(6)
1		B0 (1)	B0(1)	B0(1)	A2(4)	A2(4)	A2(4)	A3(7)	A3(7)
2			A1(2)	A1(2)	A1(2)	A1(5)	A1(5)	A1(5)	B3(8)
	FP	FP	FP		FP(R)		FP(R)	FP(R)	FP(R)

The reference instant is annotated with parenthesis. The victim is the one with smaller reference instant. In all 7 page faults, 4 of them with replacement.

b) Second chance with global replacement

	A0	B0	A1	A0	A2	A1	B0	A3	B3
marco	t=0	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8
0	A0	A0	A0	A0	A2	A2	A2	A2	A2
1		B0	B0	B0	B0	B0	B0	A3	A3
2			A1	A1	A1	A1	A1	A1	B3
	FP	FP	FP		FP(R)			FP(R)	FP(R)

The order must be kept, as well as the reference bit to identify the victim

t=3

pag	ref
A0	1
B0	1
A1	1

t=4

pag	ref
A0	0
B0	0
A1	0

t=4 → page fault.
Looking for 2nd chance victim

pag	ref
A2	1
B0	0
A1	0

t=5

pag	ref
A2	1
B0	0
A1	1

t=6

pag	ref
A0	1
B0	1
A1	1

t=7

pag	ref
A0	0
B0	0
A1	0

t=7 → page fault.
Looking for 2nd chance victim

pag	ref
A0	0
A3	1
A1	0

b) Explain if the LRU page replacement algorithm suffers of Belady's anomaly
LRU is a stack algorithm so as all stack algorithms it doesn't suffers Belady's anomaly.

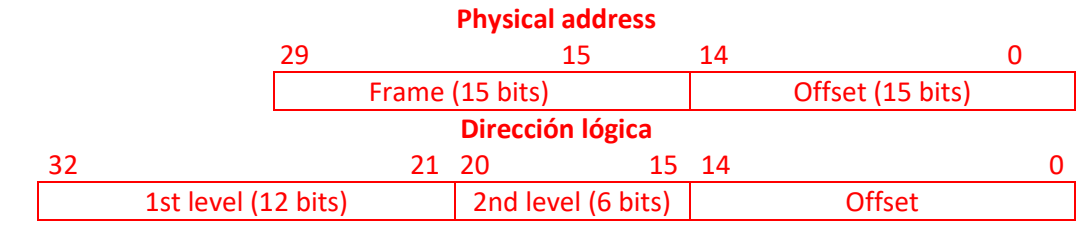
4. In a memory system with paging at two levels, the first level page table can hold up to 4096 references to page tables on the second level. The page size is 32 KByte. This system logical address space is 8GByte and it has 1GByte of physical memory.

a) Describe the structure of the logical addresses and physical addresses of this system.

b) Indicate the number of page descriptors that can contain each page table on the second level.

(0,75 points)

4 a)



- Physical space = 1GB --> 30 bit physical address
- Page size = frame size = 32KB -> 15 bit offset
- Logical space 8GB --> 33 bit logical address
- 4096 descriptors in the first level --> 2^{12} -> 12 bits
- Number of bits for 2nd level 33-(12+15) = 6

b) As there is 6 bits for page id in the 2nd page level, every table can only manage 2^6 pages, that is 64 page descriptors

---

5. Analyze the following program called expipe2.c, and assuming that no errors occur in system calls, give an explained answer to the following questions:

a) The contents of the file descriptor tables for every process when their execution goes to comments */ * (1) descriptor table(s) */* and */ * (2) descriptor table(s) */*.
b) Make a diagram of the communication scheme between processes.
c) Explain what is shown on the screen when expipe2.c is executed.

```
// expipe2.c
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    int i, fd[2];
    pipe(fd);
    for(i=0; i<2; i++) {
        if (fork() == 0) {
            dup2 (fd[0], STDIN_FILENO);
          /***(1) descriptor table(s) ***/
            close (fd[0]);
            close (fd[1]);

            execlp("/bin/cat", "cat", NULL);
            fprintf(stderr,"The exec of %s failed", argv[1]);
            exit(1);
        }
    }
    dup2 (fd[1], STDOUT_FILENO);
    close (fd[0]);
    close (fd[1]);
  /***(2) descriptor table(s) ***/
    execlp("/bin/ls", "ls", "-l", NULL);
    perror("The exec of ls failed");
    exit(1);
}
```

(1,0 point)

5

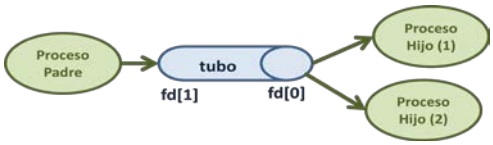
a) a) In /* (1) tabla(s) de descriptores */ two children are created, their file descriptor tables and the one for the parent process in /* (2) tabla(s) de descriptores */ are as follows:

Child 1	
0	fd[0]
1	STDOUT
2	STDERR
3	fd[0]
4	fd[1]

Child 2	
0	fd[0]
1	STDOUT
2	STDERR
3	fd[0]
4	fd[1]

Parent	
0	STDIN
1	fd[1]
2	STDERR
3	
4	

b)



b) It is shown on the screen the execution result of command: "ls –l | cat"

Proceso padre ejecuta ls –l habiendo redireccionado la salida al tubo, y los dos procesos hijos leen del tubo. Cada uno leerá parte de la salida del ls –l y la imprimirán en pantalla (orden cat). El resultado será lo mismo que ls –l sin replicar ninguna línea. En principio se debería mantener el orden de las líneas, aunque podría alterarse en función de los tamaños de los buffers que use cat.

---

6. The following program *expipe1.c* creates two processes that communicate with a pipe. Complete the program with calls and instructions at locations shown with /* *Adjust descriptors (X)* */ and replace the comment /* *descrX* */ by the appropriate variables so:

- The parent process sends to the child, through the pipe, everything that it reads in its standard input.
- The child process prints in its standard output everything it receives through the pipe.

In addition both process count the number of characters that read or write into the pipe and write a message on its standard output with this information.

Execution example: *$ echo "test" | ./expipe1*

test
We've written 4 characters
We have read 4 character

s

```
// expipe1.c
#include <string.h>
#include <stdio.h>

int main(void){
    int fildes[2];
    char ch;
    int Iread=0, Iwrite=0;
    pipe(fildes);
    if (fork()==0){
        /* Child reads in the pipe an writes in standard output */
        /** Adjust descriptors (1) **/
        while(read(/**descr1**/, &ch, sizeof(ch)) == sizeof(ch)){
            write(/**descr2**/, &ch, sizeof(ch));
            Iwrite++;
        }
        /** Adjust descriptors (2) **/
        printf("\n We have written %d caracteres\n", Iwrite);

    } else {
        /* Parent reads in standard input and writes in the pipe */
        /** Adjust descriptors (3) **/

        while (read(/*descr3*/, &ch, izeof(ch)) == sizeof(ch)){
            write(/*descr4*/, &ch, sizeof(ch));
            Iread++;
        }
        /** Adjust descriptors (4) **/
        printf("\n We have read %d characters\n", Iread);
    }
}
```

(1,0 point)

6

```
close (fildes[1]); /** Adjust descriptors (1) **/

/**descr1**/fildes[0]

/**descr2**/ 1

close (fildes[0]); /** Adjust descriptors (2) **/


-------------------------------------------------------

close(fildes[0]); /** Adjust descriptors (3) **/

/*descr3*/ 0

/*descr4*/ fildes[1]

close(fildes[1]); /**ajustad descriptores (4)**/
```

7. Given the following directory listing in a POSIX system:

```
drwxr-xr-x          2 sterrasa     fso              4096 sep  8     2012  .
drwxr-xr-x         11 sterrasa     fso              4096 dec 10     14:39 ..
-rwsrw-r-x          1 sterrasa     fso           1139706 sep  9     2012  copia
-rw-rw-r--          1 sterrasa     fso            634310 sep  9     2012  f1
-r--r--rw-          1 sterrasa     fso            104157 sep  9     2012  f3
```

Where program "copia" copies the contents of the file specified in the first argument in another specified in the second argument. Fill the table, indicating for every command in case of success which are the permissions that are checked and, in case of error, which is the permission that fails and why.

(1,0 point)

7

Notice that the executable file "copia" has the SETUID bit active:

-rwsrw-r-x 1 sterrasa fso 1139706 sep 9 2012 copia

So users that can execute "copia" will become "sterrasa" along the execution

User	Group	Command	¿Does it work?	Explanation
Inma	fso	copia f1 f2	NO	Imma cannot execute "copia" because she belongs to the same group (fso) as the owner so her permission are defined by the second permission set that doesn't contain 'x'.
Sara	ltp	copia f1 f2	YES	Sara can execute "copia". Sara is not the owner and doen't belong to "fso" group, so se belongs to "others" then her permission set is the third that contains 'x'. When she executes "copia", due to being the SETUID bit active, she becomes (sterrasa, ltp). With this (eUID, eGID) the process can read file "f1" and it can create a new entry in the working directory ('w' permission on "." in the first permission set).
Vicent	tal	copia f1 f3	NO	Vicent, as in the previous case, belongs to "others" so he can execute "copia" and along the execution he becomes (sterrasa, tal). With this (eUID, gUID), in spite of being able to read file "f1" he cannot write on "f3" (no 'w' in the first permission set) so the command fails

8. Given the following directory listing of a directory in a POSIX system, printed by the command:

```
$ ls -lia
```

```
i-node      permissions  links     UID      GID          size
9176729 drwxr-xr-x 2 silterb1 disca-upvnet 4096 2012-12-11 16:00 .
7212687 drwxr-xr-x 5 silterb1 disca-upvnet 4096 2012-12-11 15:58 ..
9176730 -rw-r--r-- 1 silterb1 disca-upvnet   25 2012-12-11 15:59 f1
9176732 lrwxrwxrwx 1 silterb1 disca-upvnet    2 2012-12-11 15:59 f2 -> f1
9176733 -rw-r--r-- 2 silterb1 disca-upvnet   32 2012-12-11 16:00 f3
9176733 -rw-r--r-- 2 silterb1 disca-upvnet   32 2012-12-11 16:00 f4
```

Explain your answer to the following questions:

a) The number of different files referenced by the entries in this directory
b) The number of links of file "."

(0,75 points)

8	a) In this directory there are a total of 6 entries, but these correspond with only 5 different i-nodes, then the number of different files referenced is 5. F3 and f4 files have the same i-nodei and therefore are the same file. File f2 is a symbolic link, this file contains a path to access f1.
	b) The minimum number of links to a directory is 2, since you can always access to it from an entry created in its parent directory and also using the "." entry inside the directory itself. So that is the case of the listed directory.

9. A disc with 8 GB of capacity, is formatted with a version of MINIX, with the following parameters:

- The boot block and the superblock occupy 1 block each.
- The i-node size is 64 bytes, with 32-bit pointers to zone (7 direct pointers, 1 indirect, 1 double indirect).
- Each directory entry is 32 bytes.
- 1 zone = 1 block = 2Kbytes
- When formating the header has reserved space for 4096 i-nodes.
- The outline of the different elements of the disk is as follows:

Boot block	Super block	i-node bitmap	Zone bitmap	i-nodes	Data area

Answer the following questions:

a) Calculate the number of blocks that occupies the i-node bitmap, the zone bitmap and the i-nodes as well as the number of data zones in the data area.

b) Suppose that in the disk exists only the root directory that contains 10 regular files, each of them of 50KBytes. Explain the number of data areas occupied in this case.

(1,25 points)

9 a)

i-node bit map

4.096 i-nodes → required 4.096 bits.

Block size 2Kbytes, every block has 16Kbits, so 1 bloque is enough for zone bit map

Disk size 8Gbytes= $8x2^{30}=2^{33}$ bytes, dividing by the zone size $2KB=2^{11}$ bytes→ $2^{33}/2^{11}=2^{22}$ zones. One bit per zone, so dividing by 16Kbits (per block) we get the number of blocks: $2^{22}/2^{14}=2^8=256$ blocks. i-nodes:

i-node size is 64 byte and there are 4096 i-nodes

Are requiered $4.096x64$ bytes=$2^{12}x2^6=2^{18}$ bytes → $2^{18}/2^{11} = 2^7$ = 128 blocks.

Data area, is the remaining space after the header 2^{22} –(1+1+1+256+128)

b)

The root directory has 10 files appart from . and .. so in all 12 directory entries. Every directory entry is 32 bytes= 12*32bytes < 2.048 bytes so it will take only one zone (one block block).

Every file requires 25 zones of 2Kbytes for data, as 25>7 it requires an indirect block so we will have 26 zonas for 10 files = 260 zonas = 260 bloques.

In all there will be 261 blocks occupied on the data area.