UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Informática de Sistemas y Computadores (DISCA)

fSO

EEE2: Ejercicio de Evaluación

12 de Enero de 2018

etsinf — Escola Tècnica Superior d'Enginyeria Informàtica

| | | | | |
|---|---|---|---|---|
| SURNAME | | NAME | | Group |
| ID | | Signature | | |

- **Keep the exam sheets stapled.**
- **Write your answer inside the reserved space.**
- **Use clear and understandable writing. Answer briefly and precisely.**
- **The exam has 7 questions, everyone has its score specified.**

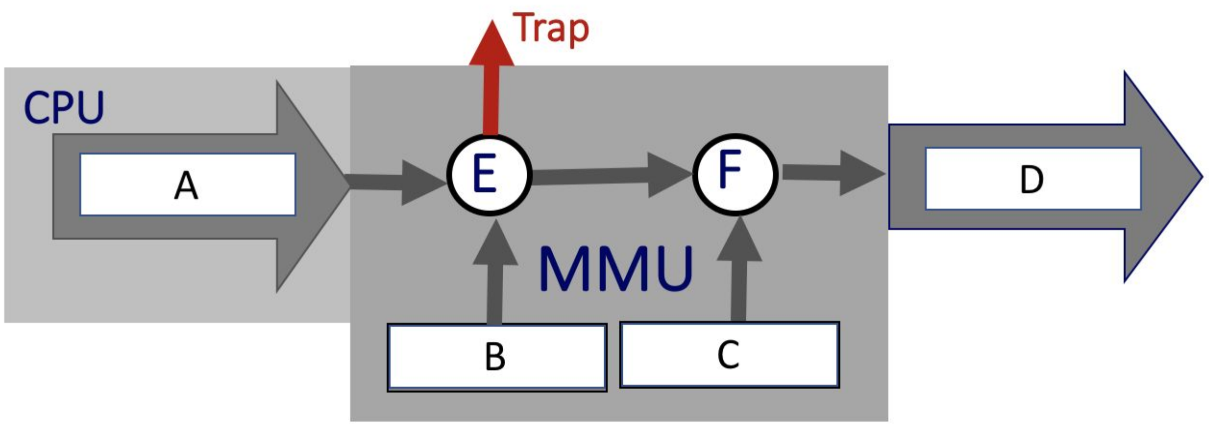
**1.** Given the following statements about the memory map of a 32-bit Linux process indicate which are true and which are false. (**Note**. An error voids a correct answer)

**(1,0 points)**

| 1 | Statement | T/F |
|---|---|---|
| | The region for initialized data can be identified by checking that it has rw-p permissions and that it has support on the executable file. | T |
| | When creating a local variable, the size or the heap region increases in order to allocate it in memory. | F |
| | When linking a library using static linking, the library information appears in a different specific region on the process memory map. | F |
| | If we show the pointer value of a variable declared inside the main function, it will be located in the region of uninitialized data . | F |
| | Using memory-mapped files improves access time to file information. | T |

**2**. The following figure represents the basic operation of an MMU (Memory Management Unit), where the name of some elements have been removed, and they have been replaced by letters from A to F. On the following table, fill column NAME with the name of the elements. Considering that the value of element A is 1000, put in column VALUES the following values: 200, 1200 and 1500, as they correspond in boxes from B to D, assuming that the issued address is correct. .

**( 1,0 points)**



**2**

| ELEMENTS | NAME | VALUES |
|---|---|---|
| A | Logical address | 1000 |
| B | Limit register | 1500 |
| C | Base register | 200 |
| D | Physical address | 1200 |
| E | < | --- |
| F | + | --- |

**3.** A memory management system, based on two level paging, allows up to 16 first level descriptors and second level tables with 256 descriptors each. The page size is 4 Kbytes and every page descriptor, both 1st and 2nd level, is 16 bytes. Addressable main memory is 16 Mbytes. Give your explained answer to the following points:

**(1.5 points = 0.5 + 0.25 + 0.75)**

**3**

**a)** Logical and physical address formats, with name and number of bits for every field or element

LA is 24-bit → offset 12 bits, 2nd level page id 8 bits and 1st level page id 4 bits.
Main memory = 16Mbytes → PA is 24-bit: offset 12 bits, frame id 12 bits

```
Logical address (name and number of bits for every element)
 ┌──────────────────────────────────────────────────────┐
 │1st level | 2nd level  |          Offset               │
 └──────────────────────────────────────────────────────┘
 b23    b20 b19       b12 b11                          b0
Physical address (name and number of bits for every element)
 ┌──────────────────────────────────────────────────────┐
 │      Frame              |             Offset          │
 └──────────────────────────────────────────────────────┘
 b23                   b12 b11                          b0
```

**b)** Obtain the hexadecimal range for logical addresses associated with the first 32 pages of a process in the system. Use the following format for your answer [First address, Last address]

[0x000000,0x01FFFF] First logical address corresponds to the top of the page 0 bringing the hexadecimal value 0x000000. Last page in the range will be requested is 31 so you have to get the equivalent value in hexadecimal (1F) and the last address that page will correspond to maximum offset. Finally we get 0x01FFFF as the last logical address in the range requested.

**c)** Explain how many 1st and 2nd level descriptors requires a process with 2K pages. Also, obtain the total number of pages occupied by the process' 2nd level tables.

Nº of 1st level descriptors:

To get the number of first level descriptors we divide 2K pages allocated to the process between the number of descriptors 2nd level (256) with the result of 8 descriptors.

Nº of 2nd level descriptors:

The number of 2nd level descriptors correspond to the 2K pages, so 2048

Nº of pages occupied by 2nd level tables:

Each of these descriptors occupies 16 bytes with what we have 32Kbytes of space required and divided by the page size (4K) we conclude that they need 8 pages.

**4.** In a system with virtual memory and demand paging there are currently running two processes: A and B. For these processes, the system has a total of 5 frames. The following table describes the evolution of frame allocation to pages demanded by both processes from time t = 120 to t = 130. Each cell details the process involved, the accessed page id (in hexadecimal) and the value of the reference bit. In the first column the starting state includes in parentheses the times for the last reference to the page.

| | **t= 120** | | **t= 121** | | **t= 122** | | **t= 123** | | **t= 124** | | **t= 125** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **↓frame / page →** | **B:d34** | | **B:d34** | | **B:c08** | | **B:c08** | | **A:002** | | **A:555** | |
| c0 | A:36a (t=25) | 1 | A:36a | 1 | A:36a | 1 | A:36a | 1 | A:36a | 1 | A:36a | 1 |
| c1 | A:450 (t=50) | 0 | A:450 | 0 | A:450 | 0 | A:450 | 0 | A:450 | 0 | A:555 | 1 |
| c2 | B:d34 | 1 | B:d34 | 1 | B:d34 | 1 | B:d34 | 1 | B:d34 | 1 | B:d34 | 1 |
| c3 | | | | | B:c08 | 1 | B:c08 | 1 | B:c08 | 1 | B:c08 | 1 |
| c4 | | | | | | | | | A:002 | 1 | A:002 | 1 |

| | **t= 126** | | **t= 127** | | **t= 128** | | **t= 129** | | **t= 130** | |
|---|---|---|---|---|---|---|---|---|---|---|
| **↓frame / page→** | **A:555** | | **B:3af** | | **B:4b0** | | **B:d00** | | **A:0fe** | |
| c0 | A:36a | 1 | A:36a | 1 | A:36a | 1 | A:36a | 1 | A:36a | 0 |
| c1 | A:555 | 1 | A:555 | 1 | A:555 | 1 | A:555 | 1 | A:555 | 0 |
| c2 | B:d34 | 1 | B:3af | 1 | B:3af | 1 | B:d00 | 1 | B:d00 | 1 |
| c3 | B:c08 | 1 | B:c08 | 0 | B:4b0 | 1 | B:4b0 | 0 | B:4b0 | 0 |
| c4 | A:002 | 1 | A:002 | 1 | A:002 | 1 | A:002 | 1 | A:0fe | 1 |

Answer the following items relying on the former information:

**(2 points = 0.5 + 0.5 + 0.5 + 0.5)**

**4**

**a)** Indicate whether the replacement algorithm used is 2nd chance or LRU and if its scope is local or global. Just in case, indicate the first event that discards one of the algorithms. Indicate also the first event that discriminates between local or global scope.

The algorithm is 2nd chance with local scope. It can not be LRU because at time t = 125 it has not been chosen as a victim the least recently accessed page, that was the A: 36a (last access at t = 25) but the A 450 that was accessed last time at t = 50. At time t = 129 it is shown that the scope is local, in case of being global it would have chosen page A: 002 as victim.

**b)** If pages accessed at instants t = 131 and t = 132 are A:36a and A:070, respectively, complete this elements as a continuation of the former table:

| **t= 131** | | **t= 132** | |
|---|---|---|---|
| **A:36a** | | **A:070** | |
| A:36a | 1 | A:36a | 0 |
| A:555 | 0 | A:070 | 1 |
| B:d00 | 1 | B:d00 | 1 |
| B:4b0 | 0 | B:4b0 | 0 |
| A:0fe | 1 | A:0fe | 1 |

**c)** Assuming that a working set policy is applied, with a window size of four, determine the working sets for processes A and B after access completion at time t = 130.

WS(A)= {002, 555, 0fe}
WS(B)= {c08, 3af, 4b0, d00}

**d)** Explain if at time t = 130 thrashing happens for processes A and B, relying on the working set criteria and assuming global frame management.

Thrashing is considered to occur if there are not enough frames to accommodate processes active areas. For processes A and B there are 5 frames avalilable, but the total working set size for A and B is WSS (A) + WSS (B) = 3 + 4 = 7 which is greater than 5, so in this conditions there is thrashing.

**5.** Given the process inheritance mechanisms in Unix and the POSIX system calls, answer the following items: :

**(1.5 points = 0.5 + 1.0)**

5

**a)** Consider that the command : `$cat < proc.c | grep “for” > loop.txt` has to run and end correctly. Draw the file descriptor tables for every process involved, describing the content of non empty descriptors. **Note.** Name the pipe as “fdpipe”

```
Process “cat” table        Process “grep” table
  0 | “proc.c”                0 | fdpipe[0]
  1 | fdpipe[1]               1 | “loop.txt”
  2 | STDERR                  2 | STDERR
```

**b)** Complete the following C program with the sentences and system calls required in order to perform the same operation as command: `$ /bin/ls | /bin/grep “.c” > my_programs`
**Note.** Consider that system calls are error free.

```
/** Prog.c:   $/bin/ls | /bin/grep “.c” > my_programs **/
#include <unistd.h>
#define newfile (O_WRONLY | O_CREAT | O_TRUNC)
#define mode644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

int main() {
    int fdpipe[2];
    int fd;

    pipe(fdpipe);

    if (fork()) {

        dup2(fdpipe[1], STDOUT_FILENO);
        close(fdpipe[0]); close(fdpipe[1]);
        execl("/bin/ls", "ls", NULL);

    } else {
        fd = open("my_programs", newfile, mode644);

        dup2(fdpipe[0], STDIN_FILENO);
        dup2(fd, STDOUT_FILENO);
        close(fdpipe[0]); close(fdpipe[1]); close(fd);
        execl("/bin/grep", "grep", ".c", NULL);
    }

    return 0;
}
```

**6.** Given the following listing of the content of directory ***XXXX/*** (with unknown name) on a POSIX system:

```
i-node   permissions links user   group   size  date   time  name
13504322 drwxrwxr-x  3    aperez  disca   4096 dic 21 19:33 .
12200756 drwx------  5    aperez  disca   4096 dic 21 19:20 ..
13504405 drwxrwxr-x  2    aperez  disca   4096 dic 21 19:29 back
13504425 -rwxr-xr-x  1    aperez  disca 151024 dic 21 19:33 cp
13504425 -rwsr-xr--  1    aperez  disca 151024 dic 21 19:33 cp2
13504417 -rw----r--  1    aperez  disca   6364 dic 21 19:22 dat1
13504421 -rw-rw-r--  3    ana     fso     1134 dic 21 19:25 dat2
13504421 -r--r--r--  3    juan    aic     1134 dic 21 19:25 dat3
```

```
./back:
13504405 drwxrwxr-x  2   aperez  disca    4096 dic 21 19:29 .
13504322 drwxrwxr-x  3   aperez  disca    4096 dic 21 19:33 ..
13504420 lrwxrwxrwx  1   manolo  disca       7 dic 21 19:29 old1 -> ../dat1
13504421 -rw-rw-r--  3   maria   fso      1134 dic 21 19:25 old2
```

**(1.5 points = 1 + 0,5)**

**6**

**a)** cp and cp2 programs are copies of command cp that copies the contents of the file received as the first argument into another file which name is indicated as the second argument. That is, "cp a b" copies the contents of the file a to file b, if b does not exist then b is created and the copy is done. Fill the table indicating whether the corresponding command works, the EUID and EGID (effective UID and GID for the process that executes the command) and in case of error, what is the missing permission(s).

| **(UID,GID)** | **COMMAND (from directory XXXX/)** | **Does it work?** | **(EUID,EGID)** | **If it fails tell the missing permission(s)** |
|---|---|---|---|---|
| (ana,fso) | cp dat1 dat4 | NO | (ana,fso) | W on XXXX/ |
| (ana, disca) | cp2 dat1 back/old3 | YES | (aperez,disca) | |
| (ana,disca) | cp back/old1 dat2 | NO | (ana,disca) | R on old1->../dat1 |
| (maria,fso) | cp dat3 .. | NO | (maria,fso) | W on .. |

**b)** After running command: `ls -li ..`
we get:

```
    i-node   permissions links user group size  date    time   name
    12200756 drwx------    5 aperez aperez  4096 dic 26 11:52 .
    10223617 drwxr-xr-x 149 aperez aperez 12288 dic 25 01:10 ..
    13764791 drwxrwxr-x    2 aperez aperez  4096 dic 26 11:51 alumnos
    13504322 drwxrwxr-x    3 aperez aperez  4096 dic 26 11:51 organizacion
    13636461 drwxrwxr-x    2 aperez aperez  4096 dic 26 11:50 practicas
```

What is the name of directory ***XXXX/***?

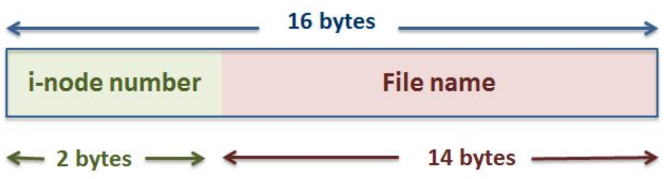
According to the initial listing, the current directory (./) has i-node 13504322. Looking which directory on the new list has assigned this node-i, we notice that it is directory: Organization

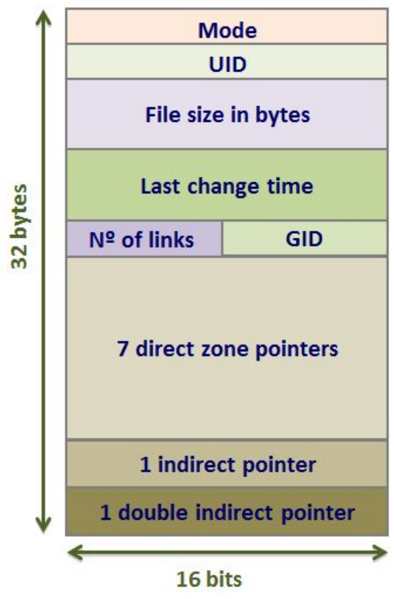
**7.** The following figures refer to the sizes and structures of the elements used to format a partition with a MINIX file system, with the following configuration :

- Nº of i-nodes : 3424
- Block size : 1 KByte
- 1 zone = 1 block
- Partition size : 10 MBytes

Notice that all the i-node fields are 16-bit, except "Nº of links" and GID that are 8-bit.

Directory entries have the following format:





**(1,5 points = 0,5 + 0,2 + 0,8)**

**a)** Number of blocks that occupy the following header elements: i-nodes bitmap, data zones bitmap and i-nodes .

```
#blocks i-nodes bit map = 3424/(1024*8) < 1 -> 1 block
#blocks zones bit map = 10*1024/(1024*8) = 10/8 -> 2 blocks
#blocks i-nodes = 3424*32/1024 = 3424/32 = 107 blocks
```

**b)** Block index for the first block in the data area .

```
Header blocks = 1 + 1 + 1 + 2 + 107 = 112 -> from 0 to 111
First block on data area: 112
```

**c)** Just after formatting the partition a 64 KByte file named "message.txt", is created. Knowing that the first block assigned to the file is 113, and that blocks are allocated in order, obtain in what block is located byte 62459 from file "message.txt"

```
A 64 KBytes file occupies 64 data areas. To address these blocks 7 direct pointers
and one indirect pointer will be used (the double indirection it is not used as the
single indirect pointer has an addresing capacity of 512 KBytes). Using the
indirect pointer means to allocate an area for direct pointers. Thus the order of
zone allocation is:

7 data zones + 1 pointers zone + 57 data zones

Byte 62459 is on relative zone: 62459/1024 = 60.995 -> 60, so it is
after the pointers zone, then it will be relative zone 60 + 1 = 61
As the first zone allocated to the file is 113, then the absolute block
in which byte 62459 is allocated is: 113 + 61 = 174
```