UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Departamento de Informática de Sistemas y Computadoras (DISCA)

fSO

EEE2

December, 22th 2014

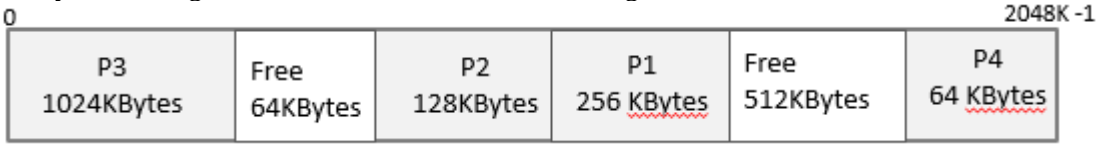
etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

SURNAME		**NAME**		Group
IDN		**Signature**		E

- **Keep the exam sheets stapled.**
- **Write your answer inside the reserved space.**
- **Use clear and understandable writing. Answer in a brief and precise way.**
- **The exam has 10 questions, everyone has its score specified.**

---

1. In a system with **contiguous allocation** memory management with **variable partitions**, the memory state at a given moment in time is the following:



a) What is the content of MMU registers that allow the translation of logical addresses sent by the CPU for processes P1, P2, P3 and P4, into physical addresses?
b) Supposing that the free gaps list is ordered from lower to higher addresses, indicate in what free gap will be allocated a new process of 28Kbyte size for every one of the allocation strategies: Best Fit, Worst Fit and First Fit.
c) Explain if in that type of system there could be external and/or internal fragmentation.

0.75 points

1

a)
In these memory management policies the MMU uses two register types: Base Register and Limit Register.
The base register contains the first physical address where the process has been allocated in main memory.
The limit register contains the process size or the last physical address (plus 1), depending on how the address check is implemented inside the MMU.
Process P1 → Base Register = 1216K, Limit Register = 1216K+256K = 1472K
Process P2 → Base Register = 1088K, Limit Register = 1088K+128K = 1216K
Process P3 → Base Register = 0, Limit Register = 1024K
Process P4 → Base Register = 1984K, Limit Register = 1984K+64K = 2048K

b)
Best Fit → 64 KByte gap
Worst Fit → 512 KByte gap
First Fit → 64 KByte gap

c)
In this type of systems it can happen external fragmentation because the taken gaps are adjusted to the process size → so there may be enough memory available to allocate a new process but because it is not contiguous the process can't be allocated → compaction techniques can be applied to put all the required memory into a single gap

2. In a system with 1 GByte of main memory and a logical space of 4GByte a memory manager with **three levels of paging** is going to be implemented. The first level tables have 16 page descriptors and the tables for 2nd and 3rd levels have the same number of page descriptors. The page size is 16Kbyte, and the page descriptor size is always 8Byte.

Explain your answer to the following questions:

a) Logical and physical address formats, indicating the composing fields and their number of bits.

b) Size of the page tables at every level and maximum number of tables at every level.

c) If a 16Mbyte process is allocated in main memory, indicate the number of page descriptors that will be used in order to address all the process' pages.

1.0 points

2

a) Logical and physical address formats

Main memory 1GByte → 30 bit physical addresses

Logical addressing space 4GByte → 32 bit logical addresses

Page size 16KByte → 14 bit offset for both pages and frames

Physical address format: 30 bit = 16 bit Frame ID + 14 bit Offset

Logical address format: 32 bit = 4 bit Page ID 1fst level + 7 bit Page ID 2nd level + 7 bit Page ID 3rd level + 14 bit Offset

b) Size of page tables for every level and maximum number of page tables at every level

1fst level page table size: 2^4 descriptors * 8Bytes =2^7 Bytes = 128Bytes

2nd level page table size: 2^7 descriptors * 8Bytes =2^{10} Bytes = 1KBytes

3rd level page table size: 2^7 descriptors * 8Bytes =2^{10} Bytes = 1KBytes

The maximum number of tables is: 1 Table 1fst level + 16 Tables 2nd level + 16*128 Tables 3rd level

c)

A 16MByte process will occupy 16MBytes/ 16Kbytes → 1024 pages → 1024 page descriptors

In the 3 level paging system the 3rd level tables can contain up to 128 page descriptors every one → so 8 tables in the 3rd level are required (8*128=1024), 1 table in the 2nd level with 8 busy page descriptors and one table in the 1fst level with only one page descriptor busy.

3. In a time sharing system that at a given moment in time is running three processes (A, B y C), a working set model has been implemented in order to avoid thrashing, with a working set window size of 5. Consider the following sequence of page references and the marked instants t_1 and t_2:

A2	B4	B3	C2	A6	C2	C1	B2	B4	B1	A2	A1	C2	B1	A4	C3
															↑ t_1
A2	B1	A3	A4	C6	B0	B5	A2	A3	B1	C7	C2	A2	A2	B1	B4
															↑ t_2

Explain if every one of the following statements are correct or false:

a) The working set of processes A, B and C at t_1 is : $WS_{t1}(A)=\{1,4\}$, $WS_{t1}(B)=\{1\}$, $WS_{t1}(C)=\{2,3\}$
b) The size of process A working set at t_2 is 3 ($WSS_{t2}(A)=3$)
c) If there are 11 frames available along all-time instants we can be sure that there won't be thrashing.

0.75 points

3

a) The working set of processes A, B and C at t_1 is : $WS_{t1}(A)=\{1,4\}$, $WS_{t1}(B)=\{1\}$, $WS_{t1}(C)=\{2,3\}$

FALSE, *to compute the working set we have to analyse the last Δ references of the three processes, then the working sets are:*

$WS_{t1}(A)=\{1,2,4,6\}$, $WS_{t1}(B)=\{1,2,3,4\}$, $WS_{t1}(C)=\{1,2,3\}$

b) The size of process A working set at t_2 is 3 ($WSS_{t2}(A)=3$)

TRUE, *because if we compute the process A working set at t2 we get:*

$WS_{t2}(A)=\{2,3,4\}$ so only three frames are required:

$WSS_{t2}(A) = 3$

c) If there are 11 frames available along all-time instants we can be sure that there won't be thrashing

TRUE, *because if we compute the working set for A, B and C at t_1 and t_2 we obtain:*

$WS_{t1}(A)=\{1,2,4,6\}$, $WS_{t1}(B)=\{1,2,3,4\}$, $WS_{t1}(c)=\{1,2,3\}$
$WS_{t2}(A)=\{2,3,4\}$, $WS_{t2}(B)=\{0,1,4,5\}$, $WS_{t2}(C)=\{2,3,6,7\}$

Then the working set sizes are:
$WSS_{t1}(A)=4$, $WSS_{t1}(B)= 4$, $WSS_{t1}(c)= 3$
$WSS_{t2}(A) = 3$, $WSS_{t2}(B)= 4$, $WSS_{t2}(C)= 4$

We conclude that in bout t1 and t1 11 frames (4+4+3) are needed to cover the reference locality of the three running processes in such a way that thrashing is avoided.

4. The OS of a given computer manages virtual memory by means of paging with pages of 4KByte size. The logical addressing space is 64MB and the physical is 1MB. Free frames are allocated globally following the increasing order of physical addresses, beginning from address 0x11000

a) Explain the format of logical and physical addresses, with their fields and number of bits per field.

b) Two processes Y and Z perform the following sequence of logical addresses (all the numbers are hexadecimal):

Y:1008C24, Y:1008C28, Y:2007143, Y:1008C2C, Y:2007157,
Z:1008C24, Z:1008C28, Z:1008C2C, Z:200A100, Z:200B012,
Y:2007158, Y:1009000, Y:1009004, Y:2007158, Y:1009008,
Z:1008C30, Z:200A017, Z:1058120, Z:1058124, Z:200B012

Obtain the reference string that corresponds to that access sequence.

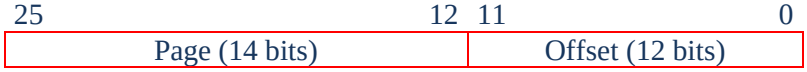
c) Obtain the content of the page tables for both processes, including only the page descriptors of referenced pages. (Suppose that there are enough free frames to allocate all the referenced pages).

(1,5 points)

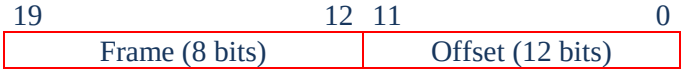
4

a) Format of logical and physical addresses, with their fields and number of bits per field

Logical address



Physical address



b) Reference string that corresponds to the access sequence

Y:1008, Y:2007, Y:1008, Y:2007,
Z:1008, Z:200A, Z:200B,
Y:2007, Y:1009, Y:2007, Y:1009,
Z:1008, Z:200A, Z:1058, Z:200B

c) Page tables for both processes:

Page Id	Page table of process Y
1008	11
2007	12
1009	16

Page Id	Page table of process Z
1008	13
200A	14
200B	15
1058	17

5. In a system that manages virtual memory by means of paging, the following reference string is generated when executing processes Y and Z (page ids in hexadecimal, left side zeros omitted): **Y:108, Y:207, Z:119, Z:20A, Y:207, Y:108, , Z:20A, Y:207, Z:20B**
In case of the system having only 3 frames available from 0 to 2, initially empty, answer the following questions:
a) Obtain the physical memory content evolution and the number of page faults when applying a replacement algorithm **LRU with global replacement**.
b) Obtain the physical memory content evolution and the number of page faults when applying a replacement algorithm **Second Chance with global replacement**.

(1,0 point)

5

a) **LRU with global replacement**. Indicate the last access time instant
Y:108, Y:207, Z:119, Z:20A, Y:207, Y:108, , Z:20A, Y:207, Z:20B

Frame									
0	Y108 (t=0)	Y108 (t=0)	Y108 (t=0)	Z20A (t=3)	Z20A (t=3)	Z20A (t=3)	Z20A (t=6)	Z20A (t=6)	Z20A (t=6)
1		Y207 (t=1)	Y207 (t=1)	Y207 (t=1)	Y207 (t=4)	Y207 (t=4)	Y207 (t=4)	Y207 (t=7)	Y207 (t=7)
2			Z119 (t=2)	Z119 (t=2)	Z119 (t=2)	Y108 (t=5)	Y108 (t=5)	Y108 (t=5)	Z20B (t=8)
	FP	FP	FP	FP(R)		FP(R)			FP(R)

The reference instant is indicated in parenthesis. When there is a page fault with memory full the victim is the allocated page that has been longer unreferenced.

Total page faults = 6
Page faults with replacement = 3

b) **Second Chance with global replacement**. Indicate bit R value and next candidate
Y:108, Y:207, Z:119, Z:20A, Y:207, Y:108, , Z:20A, Y:207, Z:20B

Frame									
0	Y108 →1	Y108 →1	Y108 →1	Z20A 1	Z20A 1	Z20A →1	Z20A →1	Z20A →1	Z20B 1
1		Y207 1	Y207 1	Y207 →0	Y207 →1	Y207 0	Y207 0	Y207 1	Y207 →0
2			Z119 1	Z119 0	Z119 0	Y108 1	Y108 1	Y108 1	Y108 0

Notation used: Pointer to next candidate (→) and reference bit

6. Suppose that the following code for *redir.c* is executed without errors and that file *"listing.txt"* doesn't exist, then answer the following questions:

a) Obtain the content of the file descriptor tables at */***(1) Descriptors table***/* and */***(2) Descriptors table***/* for every process that reaches those lines along its execution.

b) Explain what is shown on the screen whe *redir.c* code is executed and the equivalent shell commands.

c) Explain what would happen when executing *redir.c* code removing the `wait()` call from the program.

```
/****** redir.c code *******/
#include <all required.h>
#define NEWFILE (O_WRONLY | O_CREAT | O_TRUNC)
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
#define FILE "listing.txt"
int main(int argc, char *argv[]) {
  int fd;

  if (fork() == 0) {
    if ((fd = open(FILE, NEWFILE, MODE644)) == -1)
      exit(-1);
    dup2(fd, STDOUT_FILENO);
    close (fd);
    /***(1)Descriptors table ***/
    execlp("ls", "ls", "-la", NULL);
    fprintf(stderr,"Execution failed ");
    exit(1);
  }
  wait(NULL);

  if ((fd = open(FILE, O_RDONLY)) == -1)
    exit(-1);
  dup2 (fd, STDIN_FILENO);
  /***(2)Descriptors table***/
  close (fd);
  execlp("wc", "wc", NULL);
  fprintf(stderr, "wc execution failed ");
  exit(1);
}
```

(1,0 point)

6

a) The comment /*** (1) Descriptors table ***/ corresponds the the child process and the comment /*** (2) Descriptors table ***/ corresponds to the parent. The file descriptors tables are:

Child	
0	STDIN
1	fd
2	STDERR

Parent	
0	fd
1	STDOUT
2	STDERR
3	fd

b) The result is counting the number of lines, words and characters that has the listing of the directory where the command is executed.
There is no direct intercommunication between the processes, but a redirection is done to the auxiliary file "listing.txt". The following shell command will do the same:

```
$ ls –la > listado.txt
$ wc < listado.txt
```

c) The *wait* call will guaranty than the parent waits the ending of the child command (*ls*) and so the resulting file will be already created and it will be used ad the inpu of command *wc*.
If the *wait* call is removed it can happen that the parent process continues execution before the file *listing.txt* has been created and then the *open*, so the parent process would end without executing command *wc*.

---

7. The `tee` filter does a copy in the standard output of one or more files, for instance the command:

```
$ls –la | tee listing.txt
```

Shows on the screen the listing of the working directory and also copies it to file *listing.txt*.
The code (incomplete) of a command named `mitee` in shown next. This command needs: a first parameter that is the output file and one or more parameters that are interpreted as a command to be executed with its parameters. The final effect is that `mitee` executes the command specified, making a copy of the command standard output in the specified file. For instance,

```
$mitee listado.txt ls –la
```

Will do the same as executing *$ls –la | tee listing.txt*. In the code provided, the output duplication is made in function *tee_out* copying from the input on the standard output into the output file.

a) Obtain the file descriptors that correspond to tags numbered as **1** to **8** in function main.
b) Obtain the file descriptors that correspond to tags numbered as **9** to **11**.

```
#include <all required.h>
#define NEWFILE (O_WRONLY | O_CREAT | O_TRUNC)
#define MODE644 (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

void tee_out(const char *file_name) {
  int fdout,count;
  char buf[10];
  if ((fdout = open(file_name, NEWFILE, MODE644)) == -1)
    exit(-1);
  while ((count= read(**9**, buf, sizeof(buf)))>0) {
    if (write(**10**, buf, count) != count) {
      fprintf(stderr,"Write error STDOUT_FILE\n");
      exit(-1);
    }
    if (write(**11**, buf, count) != count) {
      frintf(stderr,"Write error FILE\n");
      exit(-1);
    }
  }
  close(fdout);
  exit(1);
}

int main(int argc, char *argv[]) {
  int fd[2];
  if (argc < 3) {
    fprintf(stderr, "Arguments error\n");
    exit(-1);
  }
  pipe(fd);
  if (fork() == 0) {
    dup2(**1**,**2**);
    close(**3**); close(**4**);
    tee_out(argv[1]);
  }
  dup2 (**5**,**6**);
  close(**7**); close(**8**);
  execvp(argv[2], &argv[2]);
  fprintf(stderr, "Execution failure of %s", argv[2]);
  exit(1);
}
```

(1,0 points)

7 a)

```
dup2(**1**,**2**);  dup2(fd[0], STDIN_FILENO);

close(**3**);       close(fd[0]);

close(**4**);       close(fd[1]);

dup2 (**5**,**6**);  dup2(fd[1], STDOUT_FILENO);

close(**7**);        close(fd[0]);

close(**8**);       close(fd[1]);
```

b)

```
while ((count= read(**9** STDIN_FILENO, buf, sizeof(buf)))>0) {
  if (write(**10** STDOUT_FILENO, buf, count) != count) {
    fprintf(stderr,"Write error STDOUT_FILE\n");
    exit(-1);
  }
  if (write(**11** fdout, buf, count) != count) {
    fprintf(stderr,"Write error FILE\n");
    exit(-1);
  }
}
```

8. In a Linux file system the listing of a directory content is the following:

```
total 3216
drwxr-xr-x  38 pau  admin    1292 12 dic 18:46 .
drwxr-xr-x  11 pau  admin     374  9 dic 16:51 ..
drwxr-x---  32 pau  admin    1088 11 dic 16:35 fig
-rw-r--r--   1 pau  admin    6616 12 dic 09:46 data.txt
-rwsr-xr-x   1 pau  admin     201 12 dic 09:45 reader
-rw-r-sr--   1 pau  admin     889 12 dic 09:45 writer
-rw-r--r--   2 pau  admin   44507 12 dic 09:46 log
```

Suppose that the users in the following table start a session on the system and they try to execute the commands shown also on the table, when the working directory is the one that give the former listing. Indicate the effective UID and GID when the command starts its execution and if the operating system will allow the execution and the required access to the files and directories involved in the command. *reader* and *writer* are executable files that take as argument a file name; *reader* opens the file for reading and *writer* for writing. Commands *cat* y *cp* are the common ones from the shell, with usual accessibility and permissions.

(1,0 points)

8

User	Group	Command	*Execution allowed*?	eUID	eGID	Access allowed to files and directory?
valero	*admin*	`cp data.txt fig`	***Yes***	***valero***	***admin***	***No***
pau	*admin*	`writer data.txt`	***No***			
boi	*sports*	`cat data.txt`	***Yes***	***boi***	***sports***	***Yes***
boi	*sports*	`reader data.txt`	***Yes***	***pau***	***admin***	***Yes***
boi	*sports*	`writer data.txt`	***No***			

9. Considering that the number of links of file *log* is two (2nd column in the listing) and that its size is 44507 Bytes, explain the amount of disk space that will be released when *pau* executes command:

```
$ rm log
```

(0,5 points)

9

With this command only a directory entry is removed, that is one of the hard links to *log*. Con esta orden no se elimina más que una entrada de directorio, que es uno de los dos enlaces a *log*. To remove the file data and release space from the hard disk it is required to remove the other hard link.

10. A disk with 512 MBytes of capacity is formated with a MINIX version with the following structure:

Boot block	Super block	i-node bit map	Zone bit map	i-nodes	Data area

The sizes and values used are the following:

- 32-Byte i-node: 7 direct pointers, 1 indirect and 1 double indirect
- 16-bit zone pointer
- 16-Byte directory entries
- 1 Block = 1KByte
- 1 zone = 2^3 blocks = 8KByte
- Number of i-nodes : 8.192

Answer the following items:

a) Obtain the number of blocks required for the i-node bit map, the zone bit map and the i-nodes.
b) In this system there have been created 10 directories (including the root directory), 250 regular files, 20 symbolic links and 40 hard links to already existing regular files. Explain the number of i-nodes that will be marked as busy.
c) In the root directory there are 3 directories, 10 regular files and 5 hard links to regular files. Explain the value of the field "number of links" in i-node = 1.

(1,5 points= 0,75+0,5+0,25)

10

a)

i-node bit map:

8192 i-nodes require at least 8192 bits in the i-node bit map. The block size is 1KByte so

8192 bits / 1KByte = 8*1024 bits / 1024 Bytes = 8*1024 bits / 8*1024 bits = 1 block

Zone bit map:

Disk size is 512 MBytes = 2^9*2^{20} Bytes → dividing by the zone size 8KBytes

2^9*2^{20} Bytes / 2^3*2^{10} Bytes = 2^6*2^{10} Zones = 65536 Zones

1 bit per zone is required so

65536 bits / 1 KByte = 2^6*2^{10} bit / 2^{10} Bytes = 2^6*2^{10} bits / 2^3*2^{10} bits = 2^3 = 8 blocks

i-nodes: i-nodes are 32 byte and we have 8192 i-nodes, so we need:

8192 * 32 Bytes = $2^3 * 2^{10} * 2^5 = 2^{18}$ Bytes → 2^{18} Bytes / 2^{10} Bytes = 2^8 = 256 blocks

b)

For every file in the file system one corresponding i-node will be busy.
Directories and regular files will have an i-node assigned →10 + 250 = 260 i-nodes
Hard links are directory entries that link a name with an i-node but they don't consume i-nodes.
Symbolic links are files which content is interpreted as a file system path so every symbolic link takes one i-node → 20
Total number of busy i-nodes = 10+250+20 = 280

c)

The number of links of i-node 1 is 5 that correspond to:
- Two directory entries related to i-node 1, "." y ".." located in the root directory
- Three directory entries ".." one in every child directory of the root directory