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

fSO

Ejercicio de Evaluación
17 de Diciembre de 2012

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

APELLIDOS		**NOMBRE**		Grupo
DNI		**Firma**		

- **No desgrape las hojas.**
- **Conteste exclusivamente en el espacio reservado para ello.**
- **Utilice letra clara y legible. Responda de forma breve y precisa.**
- **El examen consta de 9 cuestiones, en cada una de ellas se indica su puntuación.**

1. Sea un sistema con una memoria principal de 2Mbytes gestionada con particiones de tamaño variable y que selecciona procesos de la cola de entrada con criterio FIFO para ubicarlos en memoria. Al iniciar el sistema se carga el SO (150 KB) en las posiciones más bajas de memoria. A continuación se solicita la ejecución de 5 procesos, en el siguiente orden: P1(200KB), P2(450KB), P3(150KB), P4(250KB) y P5(650KB), entre paréntesis se indican los requerimientos de memoria. Después de finalizar P2 y P4 llegan los procesos P6(250KB), P7(150KB) y P8(300KB), en dicho orden.

a) Utilice una política Best fit (mejor hueco), e indique el estado de la memoria, ocupación y huecos, después de asignarle memoria a P8.

b) Utilice una política Worst fit (peor hueco), e indique el estado de la memoria, ocupación y huecos, después de asignarle memoria a P8.

(1.0 puntos)

1

a) Best fit

0 — 2048K

S.O. 150K	P1 200K	libre 450K	P3 150K	libre 250K	P5 650K	Libre 198K

P6(250K)

0 — 2048K

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

P7(150K)

0 — 2048K

S.O. 150K	P1 200K	libre 450K	P3 150K	P6 250K	P5 650K	P7 150K	Libre 48K

P8(300K)

0 — 2048K

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

b) Worst fit

0 — 2048K

S.O. 150K	P1 200K	libre 450K	P3 150K	libre 250K	P5 650K	Libre 198K

P6(250K)

0 — 2048K

S.O. 150K	P1 200K	P6 250K	libre 200K	P3 150K	libre 250K	P5 650K	Libre 198K

P7(150K)

0 — 2048K

S.O. 150K	P1 200K	P6 250K	libre 200K	P3 150K	P7 150K	libre 100K	P5 650K	Libre 198K

P8(300K)

El proceso P8 no puede ubicarse en memoria, ya que no hay un hueco (espacio contiguo) igual o mayor a 300K. Se podría utilizar compactación ya que la suma de los tamaños de los huecos libres 200K+100K+198K es suficiente para ubicar P8

2. Un sistema dispone de 2 GB de espacio de direccionamiento lógico, páginas de 2 KB y 1 GB de memoria física.

a) Calcule la dirección física a correspondiente a la dirección lógica 4119, para una gestión de memoria mediante paginación. Las primeras entradas de la tabla de páginas se muestra a continuación. Justifique la respuesta.

Tabla de páginas

Página	Marco	Bit de validez
0	27	v
1	8	v
2	500	v
3	0	v

b) Calcule la dirección lógica de un proceso que genera la dirección física 10288, en un esquema de segmentación paginada con 64 segmentos máximo por proceso. Las tablas de páginas para los segmentos 0, 1 y 2, de dicho proceso son:

Tabla de Páginas Segmento 0

Página	Marco	Bit Val.
0	2	v
1	800	v
2	1024	v
3	3	v

Tabla de Páginas Segmento 1

Página	Marco	Bit Val.
0	8	v
1	0	v
2	328	v
3	5	v

Tabla de Páginas Segmento 2

Página	Marco	Bit val.
0	500	v
1	21	v
2	1	v
3	82	v

(1.25puntos)

2 a) -Espacio físico de 1GB --> la dirección física ocupa 30 bits
- Tamaño de página=tamaño de marco=2KB --> 11 bits para el desplazamiento
- Espacio lógico de 2GB --> la dirección lógica ocupa 13 bits
- El número de bits para p2 se obtiene como 31-11 = 20 bits

Dirección física

29 ... 11	10 ... 0
Marco	Desplazamiento

Dirección lógica

30 ... 11	10 ... 0
Página	Desplazamiento

Dirección lógica 4119 (pagina 2, desplazamiento 23)
4119 div 2048 = 2 →pertenece a la página lógica es 2
4119 mod 2048 = 23 →desplazamiento es 23
Se consulta tabla de páginas→ el descriptor de la página 2 contiene marco 500
Dirección física: 500 * 2048 + 23 = **1.024.023**

b) -Espacio físico de 1GB --> la dirección física ocupa 30 bits
- Tamaño de página=tamaño de marco= 2KB --> 11 bits para el desplazamiento
- Espacio lógico de 2GB --> la dirección lógica ocupa 31 bits
- Nº máximo de segmentos 64 --> 6 bits para el número de segmentos
- El número de bits para páginas se obtiene de 31-11-6 = 14bits

Dirección física

29 ... 11	10 ... 0
Marco	Desplazamiento

Dirección lógica

30 ... 24	23 ... 11	10 ... 0
Segmento (6 bits)	Página (14 bits)	Desplazamiento

Dirección física 10288 (marco 5, desplazamiento 48) →10288 div 2048 = 5
10288 mod 2048 = 48
Consultamos las tablas de segmento→ el marco número 5 aparece en la tabla del segmento 1 para la página 3, por lo que la dirección lógica que ha generado la dirección física 10288 es: (1, 3*2048+ 48) = (1, 6192)

3. El sistema operativo de cierto computador gestiona la memoria virtual mediante paginación con páginas de 4 KB. Se asignan marcos libres, si los hay, por orden creciente de direcciones físicas. En un momento dado, el sistema ha de repartir 3 marcos (del 0 al 2), que inicialmente se encuentran libres, entre dos procesos nuevos A y B.

a) Aplicando un algoritmo de reemplazo LRU con reemplazo global, indique la evolución del contenido de la memoria física si se hace referencia a las siguientes páginas: A0, B0, A1, A0, A2, A1, B0, A3, B3. Diga el número de fallos de página que se producen

b) Aplicando un algoritmo de reemplazo de Segunda Oportunidad con reemplazo global, indique la evolución del contenido de la memoria física si se hace referencia a las siguientes páginas: A0, B0, A1, A0, A2, A1, B0, A3, B3. Y el número de fallos de página que se producen.

c) Justifique si el algoritmo de reemplazo LRU presenta o no la anomalía de Belady.

Justifique todas las respuestas

(2,0 puntos)

3

a) LRU con reemplazo global

	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)

Entre paréntesis se ha puesto el instante de referencia. Para elegir víctima se selecciona aquella que hace más tiempo que hemos referenciado. Total 7 Fallos de página 4 de ellos con reemplazo.

b) Segunda Oportunidad con reemplazo global

	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)

Se debe mantener el orden, así como el bit de referencia para poder buscar victimas

t=3

pag	ref
A0	1
B0	1
A1	1

t=4

pag	ref
A0	0
B0	0
A1	0

pag	ref
A2	1
B0	0
A1	0

t=4 → fallo pag.
se busca victima con 2ª op.

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

pag	ref
A0	0
A3	1
A1	0

t=7 → fallo pag.
se busca victima con 2ª op.

c)Justifique si el algoritmo de reemplazo LRU presenta la anomalía de Belady

El algoritmo LRU es un algoritmo de pila y no presenta la anomalía de Belady. Para demostrarlo trabajamos con misma secuencia del apartado b) y suponemos que tenemos asignado una marco mas (total 4 marcos).

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

El conjunto de páginas que hay en memoria en cada instante de tiempo con 3 marcos es un subconjunto, del conjunto de páginas que hay en memoria para dichos instantes de tiempo con 4 marcos. Ejemplo en t=4 con 3marcos están las páginas A0, A2, y A1, dichas páginas también están en memoria en t=4 con 4 marcos.

4. En un sistema de memoria con paginación a dos niveles, la tabla de páginas del primer nivel puede albergar hasta 4096 referencias a tablas de páginas del segundo nivel. El tamaño de página es de 32 KBytes. El espacio de direcciones lógicas de este sistema es de 8GBytes y el espacio de direcciones físicas de 1GBytes,
a) Describa la estructura de las direcciones lógicas y de las direcciones físicas de este sistema.
b) Indique el número de descriptores de páginas que puede contener cada una de las tablas de páginas de segundo nivel.

(0,75)

4

a)

Dirección física

29 — 15	14 — 0
Marco (15 bits)	Desplazamiento (15 bits)

Dirección lógica

32 — 21	20 — 15	14 — 0
Primer nivel (12 bits)	2º nivel (6 bits)	Desplazamiento

- Espacio físico= 1GB --> la dirección física ocupa 30 bits
- Tamaño de página=tamaño de marco= 32KB->15 bits para desplazamiento
- Espacio lógico de 8GB --> la dirección lógica ocupa 33 bits
- 4096 descriptores del Primer nivel --> 2^{12}→ se necesitan 12 bits
- El número de bits del segundo nivel se obtiene 33-(12+15) = 6

b)
Como hay 6 bits para el número de página de segundo nivel, esto hace que cada una de las tablas sólo pueda acceder a 2^6 referencias a páginas, es decir, 64 descriptores

5. Analice el siguiente programa denominado *ejtubo2.c*, y suponiendo que no se producen errores al ejecutar las llamadas, indique de forma justificada:

a) El contenido de las tablas de descriptores en el punto donde está los comentarios */***(1) tabla(s) de descriptores ***/* y /*** (2) tabla(s) de descriptores ***/ para cada uno de los procesos que alcancen durante su ejecución dicha la línea.
b) Realice un diagrama del esquema de comunicación que llevan a cabo los procesos al ejecutarlo.
c) Indique de forma justificada lo que se muestra por pantalla al ejecutar dicho código.

```
#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) tabla(s) de descriptores ***/
      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) tabla(s) de descriptores ***/
execlp("/bin/ls", "ls", "-l", NULL);
perror("The exec of ls failed");
exit(1);
}
```

(1,0 puntos)

5

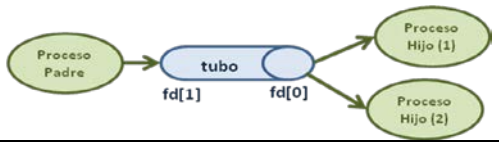
a) a) Se crean dos procesos hijos que en el punto /* (1) tabla(s) de descriptores */ el contenido de sus tablas de descriptores, así como la del proceso padre en el punto /* (2) tabla(s) de descriptores */ tendrán los siguientes descriptores:

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

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

Proceso Padre	
0	STDIN
1	fd[1]
2	STDERR
3	
4	

b)



b) Se muestra por pantalla el resultado de ejecutar la orden "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. El siguiente programa *ejtubo1.c* crea dos procesos que se comunican con un tubo. Complete el programa con llamadas e instrucciones en los lugares indicados con /**ajustad descriptores (X)**/ y sustituya los comentarios /**descrX**/ por las variables adecuadas de manera que:

- El proceso padre le transmite al hijo, por el tubo, todo lo que lee por su entrada estándar.
- El proceso hijo imprima por su salida estándar todo lo que le recibe por el tubo.

Además ambos proceso cuentan el número de caracteres que leen o escriben en el tubo y escriben un mensaje en su salida estándar con esta información.

Ejemplo de ejecución: *$ echo "prueba" | ./ejtubo1*

prueba
hemos escrito 7 caracteres
hemos leido 7 caracteres

```
#include <string.h>
#include <stdio.h>
int main(void){
int tubo[2];
char letra;
int leo=0, escribo=0;
pipe(tubo);
if (fork()==0){
   /* hijo lee del tubo y escribe en salida estándar */
   close (tubo[1]); /**ajustad descriptores (1)**/
   while(read(/**descr1**/tubo[0],&letra,sizeof(letra))==sizeof(letra)){
      write(/**descr2**/ 1,&letra,sizeof(letra));
      escribo++;
   }
   close (tubo[0]);/**ajustad descriptores (2)**/

   printf("\nhemos escrito %d caracteres\n",escribo);

}else{
   /* padre lee de la entrada estándar y lo escribe en el tubo */
   close(tubo[0]);/**ajustad descriptores (3)**/
```

```
    while (read(/*descr3*/ 0,&letra,sizeof(letra))==sizeof(letra)){
       write(/*descr4*/ tubo[1],&letra,sizeof(letra));
       leo++;
    }
    close(tubo[1]); /**ajustad descriptores (4)**/
    printf("\nhemos leido %d caracteres\n",leo);
}
}
```

(1,0 puntos)

6

Proceso padre le transmite al hijo, por el tubo, todo lo que lee por su entrada estándar
-- Proceso padre lee de la entrada estándar, en el descriptor 0
-- Proceso padre escribe en el tubo que comparten → descriptor tubo[1]

```
   while (read(/*descr3*/ 0,&letra,sizeof(letra))==sizeof(letra)){
      write(/*descr4*/ tubo[1],&letra,sizeof(letra));
      leo++;
   }
```

Proceso hijo imprime por su salida estándar todo lo que le llegue por el tubo
--Proceso hijo escribe en la salida estándar, en el descriptor 1
--Proceso hijo lee del tubo que comparte con su padre → descriptor tubo[0]

```
while(read(/**descr1**/tubo[0],&letra,sizeof(letra))==sizeof(letra)){
      write(/**descr2**/ 1,&letra,sizeof(letra));
      escribo++;
   }
```

Tanto padre como hijo están leyendo con un bucle "while()" y utilizando los descriptores del tubo. Por lo tanto para que llegue la señal SIGPIPE y finalicen los procesos, deben cerrarse los descriptores asociados a dichos tubos que no se necesiten en el bucle "while()",antes de ejecutarlo.

Proceso hijo →close (tubo[1]); /ajustad descriptores (1)**/**
Proceso padre→close(tubo[0]);/ajustad descriptores (3)**/**

Despues del while ambos deben cerrar el descriptor del tubo que todavía conservaban para que finalicen todos los procesos que se encuentren pendientes asociados al tubo

Proceso hijo→close (tubo[0]);/ajustad descriptores (2)**/**
Proceso padre-> close(tubo[1]); /ajustad descriptores (4)**/**

---

7. Dado el siguiente listado de un directorio en un sistema POSIX:

```
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
```

Donde el programa `copia` copia el contenido del archivo que recibe como primer argumento en otro cuyo nombre es el nombre recibido como segundo argumento. Rellene la tabla, indicando en caso de éxito cuales son los permisos que se van comprobando y, en caso de error, cuál es el permiso que falla y porqué.

(1,0 puntos)

7 Se ha de tener en cuenta que el programa "copia" tiene el bit de SETUID activo

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

y por tanto aquellos que puedan llegar a ejecutar copia, actuarán como sterrasa mientras dure la ejecución.

Usuario	Grupo	Orden	¿Funciona?	Justificación
Inma	fso	copia f1 f2	NO	
Imma no puede ejecutar copia. Al pertenecer Imma al grupo del usuario propietario (fso) le corresponden los permisos de la segunda tripleta y en ella no hay permisos de ejecución(x)				
Sara	ltp	copia f1 f2	SI	
Sara puede ejecutar "copia". Sara no es el propietario de copia ni pertenece al grupo fso. Sara pertenece a "others" le corresponde la tercera tripleta de permisos. Cuando ejecuta "copia", como está activo el bit de setuid, pasa a ser (sterrasa, ltp). Con esta identidad puede leer el fichero f1 y crear una nueva entrada en el directorio actual (permiso de escritura sobre la entrada ".")				
Vicent	tal	copia f1 f3	NO	
Vicent, igual que en el caso anterior, tiene permisos para ejecutar "copia" y durante la ejecución del mismo pasa a ser (sterrasa, tal). Con esta identidad, a pesar de poder leer el fichero f1, no puede escribir en f3, por lo que la orden falla				

---

8. Dado el siguiente listado de un directorio en un sistema POSIX, obtenido después de un ejecutar la orden: `$ls -lia`:

```
Nodo-i      permisos   enlaces    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
```

Indique de forma justificada:

a) El número de archivos diferentes a los que se hace referencia con las entradas de este directorio
b) El numero de enlaces del archivo "."

(0,75 puntos)

8

a)
En este directorio hay un total de 6 entradas, pero estas se corresponden con únicamente 5 nodos-i diferentes. El número de archivos distintos a los que se hace referencia son 5.
Los archivos f3 y f4 tienen el mismo nodo-i y por tanto son el mismo archivo
El archivo f2 es de tipo enlace simbólico, este archivo contiene una ruta para acceder a f1.

b)
El número de enlaces mínimo para un directorio es 2, ya que siempre se puede acceder a él desde una entrada creada en su directorio padre y utilizando el "." dentro de él. Este es el caso del directorio listado.

---

9. Un disco con una capacidad de 8GB, se formatea con una versión de MINIX , cuyos tamaños son los siguientes:

- El bloque de arranque y el superbloque ocupan 1 bloque cada uno.
- El tamaño del nodo-i es de 64 bytes, con punteros a zona de 32 bits (7 punteros directos, 1 indirecto,1 doble indirecto) .
- Cada entrada de directorio ocupa 32 bytes.
- 1 zona = 1 bloque = 2Kbytes
- Al formatear se ha reservado espacio en la cabecera para para 4.096 nodos-i
- El esquema de los diferentes elementos del disco es el siguiente

Arranque	Super bloque	Mapa de bits Nodos-i	Mapa de bits Zonas	Nodos- i	Zonas de datos

Se pide:

a) Calcule el número de bloques que ocupa el Mapa de bits nodos-i, el Mapa de bits Zonas y los Nodos-i, así como el número de Zonas de datos.

b) Suponga que en este disco existe únicamente el directorio raíz que contiene 10 archivos regulares, cada uno de ellos de 50KBytes e indique de forma justificada el número de zonas de datos ocupadas para este caso.

(1,25 puntos)

9

a) **Mapa de bits de nodos-i**:

Tenemos 4.096 nodos-i →necesitamos 4.096 bits.

Bloques de 2Kbytes, cada bloque contiene 16Kbits, luego con 1 bloque es suficiente.

Mapa de bits de zonas:

Disco de 8Gbytes= $8x2^{30}=2^{33}$ bytes, si dividimos por el tamaño de la zona que son $2KB=2^{11}$ bytes→ $2^{33}/2^{11}=2^{22}$ zonas. Necesitamos un bit por zona, por lo que dividiendo por 16Kbits (por bloque) obtendremos el número de bloques: $2^{22}/2^{14}=2^{8}=$256 bloques.

Nodos-i:

Los nodos-i son de 64 byte y tse ha formateado con 4096 nodos-i.

Serán necesarios 4.096x64 bytes=$2^{12}x2^{6}=2^{18}$ bytes → $2^{18}/2^{11} = 2^{7} =$ 128 bloques.

Zona de datos, la zona de datos ocupara todos los bloques que no ocupe la cabecera, en este caso 2^{22} –(1+1+1+256+128)

b) El directorio raíz tiene sólo 10 archivos además del . y .. por lo que tendrá 12 entradas de directorio. Cada entrada de directorio ocupa 32 bytes= 12*32bytes < 2.048 bytes por lo que ocupa una sola zona (un bloque).

Cada archivo requiere 25 zonas de 2Kbytes para datos, como 25>7 requerirá un bloque indirecto, por lo que tendremos 26 zonas por 10 archivos = 260 zonas = 260 bloques.

Total habrá 261 bloques ocupados en la zona de datos.