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

fSO

Ejercicio de Evaluación EE2
18 de Enero 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, la 4 vale 2 (0.5+1+0.5) puntos y el resto 1 punto cada una.**

Un sistema informático cuyo sistema operativo ocupa 16KBytes está dotado de una memoria física de 128KBytes. La memoria está gestionada con particiones fijas de tamaño 2KBytes.

a) Indique razonadamente, el grado máximo de multiprogramación del sistema

b) En un instante dado, están ubicados en memoria física el sistema operativo (en las posiciones más altas de memoria) y los procesos P1 y P2, estando el resto de la memoria libre. Para P1 el contenido sus registros base y límite es (4096, 1000) y para P2 es (8192, 950). Indique a qué procesos corresponden las siguientes direcciones físicas: 10500, 4150, 8250, 128000.

1

a)

Quedan 128-16=112K para procesos, y como las particiones son fijas de 2K, el número máximo de procesos que pueden alojarse en memoria de forma contigua es 112K/2K=56

b)

El rango de direcciones del proceso P1 sera de la 4096 a la 5095 y para P2 de 8192 a 9141. Lo que se salga de esos rangos o estará libre o corresponderá al sistema operativo a partir de la dirección física 112*1024. Por tanto el primero corresponde a una zona libre, el segundo a P1, el tercero a P2 y el cuarto al S.O.

Sea un sistema **SIN memoria virtual** con **segmentación paginada**, y las siguientes características:

- Direcciones lógicas de 16 bits
- Memoria física de 128 marcos
- Páginas de 4K palabras
- Un máximo de 4 segmentos por proceso
- Dado un proceso ubicado en memoria que tiene 3 segmentos: S0, S1 y S2

Segmento	Páginas completas ocupadas	Marcos donde se ubican las página
S0	4	Marco 3, marco 20, marco 63, marco 89
S1	3	Marco 16, marco 42, marco 56
S2	2	Marco 38, marco 25

a) Indique el formato de la dirección lógica, identificando sus campos y número de bits para cada uno de ellos.

b) Indique el contenido de los campos límite de la tabla de segmentos del proceso ubicado.

c) Indique las direcciones físicas que generan la MMU a partir de las siguientes direcciones lógicas (en binario) emitidas durante la ejecución del proceso:

- 0110000000000100, 1011000010000011

2

a) Segmento: 2 bits, Página: 2 bits, Desplazamiento: 12 bits

b)

S0: 11111111111111 = 4 *4KBytes
S1: 10111111111111 = 3 *4KBytes

	S2: 01111111111111 = 2 * 4KBytes
	c) 0110000000000100 -> S1, pag 2 -> marco 56 = 111000 -> despl 000000000100 Dirección física: 0111000000000001000 1011000010000011 -> S2, pag 3 -> error ya que el segmento 2 sólo tiene dos páginas

Diga si son verdaderas (V) o falsas (F) las siguientes sentencias sobre hiperpaginación:

3	Sentencias	V/F
	En un modelo de prevención de hiperpaginación mediante control de la tasa de fallos si un proceso sobrepasa el limite superior se le quitan marcos	F
	En un modelo de prevención de hiperpaginación mediante control de la tasa de fallos si un proceso tiene una tasa de fallos por debajo del límite inferior se le quitan marcos	V
	La hiperpaginación se produce cuando la suma de los tamaños de las localidades de los procesos es superior al espacio de direccionamiento lógico	F
	La hiperpaginación solo se puede dar en los sistemas que gestionan memoria virtual	V
	Los sistemas de memoria que utilizan reserva de marcos previenen la hiperpaginación	V
	El planificador a medio plazo puede actuar pasando procesos al área de swap en el caso de detectar una situación de hiperpaginación.	V

Sea un sistema informático con una memoria física de 16MBytes, paginación por demanda, direcciones lógicas de 16 bits y tamaño de página de 256Bytes. La gestión de memoria lógica se basa en paginación multinivel con dos niveles y una tabla de páginas de primer nivel de 16 entradas.

a) Determine el formato de las direcciones lógicas y físicas.

b) Dado el siguiente conjunto de referencias realizados por los procesos A y B:
(A,0x01EF),(A,0x01DF),(B,0x0213),(B,0x0302),(B,0x0489),
(A,0x01FF),(B,0x0500), (A,0x03AB),(B,0x0304),(A,0x0207),(B,0x01AA)
Represente la evolución de la memoria asumiendo que los procesos A y B solo pueden utilizar los marcos 0, 1, 2, 3 y 4, que inicialmente están libres. El algoritmo de reemplazo es LRU global.

c) Determine la dirección física correspondiente a la dirección lógica (A,0x0145)

4

a)
Direcciones lógicas: 16 bits-> 4 bits tabla pag 1 nivel + 4 bits tabla pag 2º nivel + 8 bits desplazamiento
Direcciones físicas: 24 bits -> 16 nº de marco + 8 desplazamiento

b) De las referencias eliminamos las repetidas y dejamos solo los bits de página, con lo que obtenemos la serie de referencias:
(A,0x01),(B,0x02),(B,0x03),(B,0x04),(A,0x01),(B,0x05),(A,0x03),
(B,0x03),(A,0x02),(B,0x01)

Marco	Referencias a páginas									
	A,1	B,2	B,3	(B,4)	(A,1)	(B,5)	(A,3)	(B,3)	(A,2)	(B,1)
0	**A,1**	**A,1**	**A,1**	**(A,1)**	(A,1)	(A,1)	(A,1)	(A,1)	(A,1)	**(B,1)**
1		**B,2**	**B,2**	**(B,2)**	(B,2)	(B,2)	**(A,3)**	(A,3)	(A,3)	(A,3)
2			**B,3**	**(B,3)**	(B,3)	(B,3)	(B,3)	(B,3)	(B,3)	(B,3)
3				**(B,4)**	(B,4)	(B 4)	(B,4)	(B,4)	**(A,2)**	(A,2)
4						**(B,5)**	(B,5)	(B,5)	(B,5)	(B,5)

Todas las referencias pertenecen a **la página 0 de primer nivel**
Total 8 Fallos de Página (5 fallos sin reemplazo + 3 con reemplazo)

	c) Se trata de la página de (A,0x01) que no está en memoria → se generaría fallo de página ya que se trata de una página que pertenece al proceso→la victima según LRU sería (B,0x05) en el marco 4, luego Dirección física=0x000445

A continuación se muestra el mapa de memoria de un proceso cuyo archivo binario se llama a.out:

```
0044c000-005a6000 r-xp 00000000 08:01 787275     /lib/i386-linux-gnu/libc-2.13.so
005a6000-005a7000 ---p 0015a000 08:01 787275     /lib/i386-linux-gnu/libc-2.13.so
005a7000-005a9000 r--p 0015a000 08:01 787275     /lib/i386-linux-gnu/libc-2.13.so
005a9000-005aa000 rw-p 0015c000 08:01 787275     /lib/i386-linux-gnu/libc-2.13.so
005aa000-005ad000 rw-p 00000000 00:00 0
006d6000-006d7000 r-xp 00000000 00:00 0          [vdso]
0095e000-0097a000 r-xp 00000000 08:01 787262     /lib/i386-linux-gnu/ld-2.13.so
0097a000-0097b000 r--p 0001b000 08:01 787262     /lib/i386-linux-gnu/ld-2.13.so
0097b000-0097c000 rw-p 0001c000 08:01 787262     /lib/i386-linux-gnu/ld-2.13.so
08048000-08049000 r-xp 00000000 08:01 415026     /home/patricia/a.out
08049000-0804a000 r--p 00000000 08:01 415026     /home/patricia/a.out
0804a000-0804b000 rw-p 00001000 08:01 415026     /home/patricia/a.out
0804b000-0804c000 rw-p 00000000 00:00 0
0a02a000-0a04b000 rw-p 00000000 00:00 0          [heap]
b77fc000-b77fd000 rw-p 00000000 00:00 0
b780b000-b780c000 r--s 00000000 08:01 414986     /home/patricia/prueba.c
b780c000-b780f000 rw-p 00000000 00:00 0
bfe97000-bfeb8000 rw-p 00000000 00:00 0          [stack]
```

Conteste y justifique:

a) Qué rangos de direcciones corresponden a código.
b) ¿Hay ficheros proyectados en memoria? ¿En qué rango?
c) ¿Por qué la última región etiquetada como *stack* no está respaldada por un archivo?

5	a) Todas aquellas regiones que tienen permisos de ejecución en deben corresponder a código. 0044c000-005a6000 r-xp → corresponde a código de librerías 006d6000-006d7000 r-xp→ corresponde a código de librearías 0095e000-0097a000 r-xp → corresponde a código del sistema operativo 08048000-08049000 corresponde a código del proceso en ejecución, ya que es la zona con permiso de ejecución con el nombre del ejecutable
	b) El fichero proyectado es b780b000-b780c000 r—s 00000000 08:01 414986 /home/patricia/prueba.c
	c) Se trata de la región de la pila. La pila no tiene valores iniciales y por tanto no necesita estar soportada por ningún archivo

En un sistema se ha formateado una partición con el sistema de archivos MINIX. La partición consta de 19820 bloques y tiene un número máximo de ficheros de 1230. Describa en qué partes se divide la cabecera de la partición cuántos bloques ocupa cada una de ellas. De los 19820 bloques ¿Qué porcentaje está destinado a datos de usuario? ¿Y cuanto está destinado a mantener información del propio sistema de archivos?

NOTA: Los tamaños estándar de Minix son los siguientes: nodos-i de 32 bytes (7 punteros directos, 1 indirecto,1 doble indirecto), entradas de directorio de 16 bytes, 1 zona=1 bloque=1024 bytes, puntero a zona de 16 bits.

6

1	1	1	3	39	19774
Arranque	Superbloque	Mapa-nodos-i	Mapa-zonas	Nodos-i	Datos
0	1	2	4	7	46

Porcentaje de datos de usuario → 19774/19820 = 99%
Porcentaje de datos del stma. de archivos → 46/19820 = 1%

Un disco duro está organizado en bloques de 512 Bytes. Cada índice a bloque ocupa 32 bits.

a) Si el sistema de archivos asigna una lista enlazada de bloques a cada archivo, ¿cuántos bloques ocupará un archivo de 1 Byte de datos? ¿Y un archivo de 512 Bytes?

b) Si el sistema de archivos gestiona la asignación de bloques de datos con una lista enlazada de bloques de índices, ¿cuántos bloques (tanto de índices como de datos) ocupará un archivo de 1 Byte? ¿Y un archivo de 1000 Bytes?

Justifique sus respuestas

7

a) Cada bloque de datos contendrá un índice de 4 bytes que apuntará al bloque siguiente; por lo tanto, quedarán 508 bytes par el contenido del archivo.
Un archivo de un byte ocupará un bloque. Un archivo de 512 bytes ocupará dos bloques.

b) El sistema de archivos mantendrá dos tipos de bloque: bloques de índices y bloques de contenido. En cada bloque de índices caben 512/4=128 punteros, 127 de los cuales harán referencia a bloques de datos y el otro indicará el bloque de índices siguientes. Por lo tanto:
Un archivo de un byte ocupará dos bloques: un bloque de índices (con sólo una entrada útil) y un bloque de datos (con sólo un byte relevante).
Un archivo de 1000 bytes ocupará tres bloques: un bloque de índices (con dos entradas útiles) y dos bloques de datos: el primero completo y el segundo con 1000-512=488 bytes ocupados

Dado el siguiente código en el cual se generan al menos tres procesos P1, P2 y P3:

```
...
pipe(fd);/*pipe1*/
pipe(fd2);/*pipe2*/
if(fork() != 0){
  /***Proceso P1 ***/
  dup2(fd[1],STDOUT_FILENO);
  close(fd[0]); close(fd[1]);
  dup2(fd2[0],STDIN_FILENO);
  close(fd2[0]);
close(fd2[1]);
  /*tabla proceso P1*/
}else{
  /***Proceso P2 ***/
  dup2(fd[0],STDIN_FILENO);
  close(fd[0]); close(fd[1]);
  pipe(fd); /*pipe3*/
  if(fork() != 0){
    close(fd2[0]);
    close(fd2[1]);
    dup2(fd[1],STDOUT_FILENO);
    close(fd[0]);
    close(fd[1]);
    /*tabla proceso P2*/
  }else{
     /***Proceso P3 ***/
     dup2(fd[0],STDIN_FILENO);
     close(fd[0]);
     close(fd[1]);
     dup2(fd2[1],STDOUT_FILENO);
     close(fd2[0]);
     close(fd2[1]);
    /*tabla proceso P3*/
  }
}
...
```

a) Indique el contenido de las tablas de descriptores de archivo para los procesos P1, P2 y P3, en los puntos del código marcados como /*tabla proceso Pi*/.

b) Determine el parentesco que existe entre P1, P2 y P3 así como el esquema de redirecciones resultante de ejecutar el código.

8

Proceso P1	
0	fd2[0]
1	fd[1]
2	stderr

/*pipe2*/
/*pipe1*/

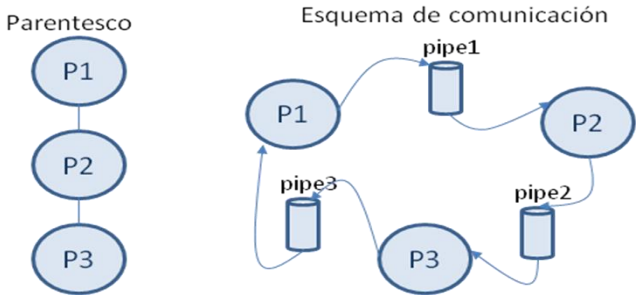
Proceso P2	
0	fd[0]
1	fd[1]
2	stderr

/*pipe1*/
/*pipe3*/

Proceso P3	
0	fd[0]
1	fd2[1]
2	stderr

/*pipe3*/
/*pipe2*/

El proceso P1 es el padre de P2, P2 es el padre de P3
Se establece un anillo de comunicación entre los tres procesos utilizando tres tubos



Sea el siguiente listado de un sistema de archivos UNIX

```
drwxr-xrwx   2    pat fso 4096        may 8 2002 .
drwxr-xrwx   11   pat fso 4096        mar 21 14:39 ..
-rwxrwxr-x   1    pat fso 1139706     abr 12 2003 copia1
-rwxrwsr-x   1    pat fso 1128236     abr 12 2003 copia2
-rw-r—r--    1    pat fso 634310      abr 9 2002 fich1
-r--r--rw-   1    pat fso 104157      abr 9 2002 fich3
```

donde los programas "copia1" y "copia2" copian el archivo recibido como primer argumento, dándole a la copia el nombre recibido como segundo argumento. Explique detalladamente si las siguientes órdenes podrán completarse con éxito o no, asuma que se ejecutan en el directorio listado anteriormente.

a) `$copia1 fich1 fich2 y $copia2 fich1 fich3`, con los atributos eUID=maria y eGID=fso, inicialmente.

b) `$copia1 fich1 fich3 y $copia2 fich1 fich2`, con los atributos eUID=anna y eGID=etc, inicialmente.

9

a) `$copia1 fich1 fich2` → **maria** de **Fso** puede ejecutar `copia1`, puede leer el fichero `fich1`, pero **no** tiene **permisos de escritura** en el **directorio actual**, por lo que **la orden falla**

`$copia2 fich1 fich3` → **maria** de **fso** puede ejecutar `copia2` y al tener el fichero el bit de SETGID activo pasa a tener un eGID=fso (que ya tenía), Siendo del mismo grupo que el propietario del fichero ejecutable, puede leer el fichero `fich1` pero **la orden falla** por **no tener permisos de escritura sobre** `fich3`

b) `$copia1 fich1 fich3` → **anna** de **etc** puede ejecutar `copia1`, puede leer el fichero `fich1`, y permisos de escritura sobre fich3, por lo que **la orden se ejecuta con éxito**

`$copia2 fich1 fich2`→ **anna** de **etc** puede ejecutar `copia2` y al tener el fichero el bit de SETGID activo pasa a tener un eGID=fso., Siendo del mismo grupo que el propietario del fichero ejecutable, puede leer el fichero `fich1` pero **la orden falla** por **no tener permisos de escritura en el directorio actual**