



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 ejercicio consta de 8 cuestiones, la cuestión 1 vale 1.5 puntos y la 3 vale 2.5 (0.75+ 1+ 0.5+0.25) puntos, el resto de cuestiones valen 1 punto cada una.
- Tiempo de realización 2 horas

Sea un sistema de memoria con particiones de tamaño variable y el siguiente estado de ocupación:

Libre 20K	Ocupado	Libre 10K	Ocupado	Libre 5K	Ocupado	Libre 15K	Ocupado	Libre 25K
--------------	---------	--------------	---------	-------------	---------	--------------	---------	--------------

donde la lista de huecos libres se mantiene ordenada hacia direcciones crecientes:

lista de huecos → 20K → 10K → 5K → 15K → 25K

A continuación solicitan ser ubicados en memoria cuatro procesos en el siguiente orden P1, P2, P3 y P4. Teniendo en cuenta que sus tamaños son P1(10K), P2(15K), P3(7K) y P4(14K), indique el estado de la memoria (huecos libres y ocupados) así como la lista de huecos si se aplican las técnicas:

- Best fit (mejor hueco)
- Worst fit (peor hueco)
- First fit (primer hueco)

1.5 puntos

<p><i>a) Best fit y lista de huecos</i></p> <p><i>lista de huecos → 13K → 5K → 11K</i></p> <table border="1"> <tr> <td>P3/ libre(13K)</td> <td>Ocupado</td> <td>P1</td> <td>Ocupado</td> <td>Libre 5K</td> <td>Ocupado</td> <td>P2</td> <td>Ocupado</td> <td colspan="2">P4/libre(11K)</td> </tr> </table>										P3/ libre(13K)	Ocupado	P1	Ocupado	Libre 5K	Ocupado	P2	Ocupado	P4/libre(11K)	
P3/ libre(13K)	Ocupado	P1	Ocupado	Libre 5K	Ocupado	P2	Ocupado	P4/libre(11K)											
<p><i>b) Worst fit y lista de huecos</i></p> <p><i>Lista de huecos → 5K → 10K → 5K → 8K → 1K</i></p> <table border="1"> <tr> <td>P2/libre(5 K)</td> <td>Ocupado</td> <td>Libre</td> <td>Ocupado</td> <td>Libre 5K</td> <td>Ocupado</td> <td>P3/libre(8K)</td> <td>Ocupado</td> <td colspan="2">P1/P4/libre (1K)</td> </tr> </table>										P2/libre(5 K)	Ocupado	Libre	Ocupado	Libre 5K	Ocupado	P3/libre(8K)	Ocupado	P1/P4/libre (1K)	
P2/libre(5 K)	Ocupado	Libre	Ocupado	Libre 5K	Ocupado	P3/libre(8K)	Ocupado	P1/P4/libre (1K)											
<p><i>c) First fit y lista de huecos</i></p> <p><i>Lista de huecos 3K → 10K → 5K → 11K</i></p> <table border="1"> <tr> <td>P1/P3/libr e 3K</td> <td>Ocupado</td> <td>Libre 10K</td> <td>Ocupado</td> <td>Libre 5K</td> <td>Ocupado</td> <td>P2</td> <td>Ocupado</td> <td colspan="2">P4/Libre 11K</td> </tr> </table>										P1/P3/libr e 3K	Ocupado	Libre 10K	Ocupado	Libre 5K	Ocupado	P2	Ocupado	P4/Libre 11K	
P1/P3/libr e 3K	Ocupado	Libre 10K	Ocupado	Libre 5K	Ocupado	P2	Ocupado	P4/Libre 11K											



Para un sistema de memoria paginado con direcciones lógicas de 22 bits y un tamaño de página de 2 Kbytes. Conteste de forma justificada:

- Indique de forma justificada cuál será el número máximo de páginas que podrá tener asignadas un proceso en este sistema.
- Si empleamos paginación a dos niveles, donde la tabla de páginas primer nivel contiene únicamente 8 entradas, ¿cuántas tablas de segundo nivel son necesarias para un proceso que requiere 1024 páginas? Razone su respuesta

1.0 puntos

2	<p>a)</p> <p>Con tamaños de página de 2Kbytes $= 2^{11}$ bytes. Son necesarios 11 bits para indicar el desplazamiento de la página y 11 bits para el número de página \rightarrow 2048 páginas</p> <p>b)</p> <p>Con 8 entradas de primer nivel son necesarios 3 bits en la dirección lógica, con lo que quedan 8 bits en el segundo nivel que son 256 entradas.</p> <p>$1024 \text{ páginas} / 256 \text{ entradas} = 4 \text{ tablas de páginas de } 2^\circ \text{ nivel}$</p> <p>Dirección lógica:</p> <p>--1er nivel (3bits)---2 nivel (8bits)--- Desplazamiento (11bits)---</p>
---	---

Sea un sistema de memoria virtual con segmentación paginada y las siguientes características:

- Direcciones lógicas de 16 bits
- Memoria física de 1024 marcos
- A cada proceso se le asigna un máximo de 4 marcos
- Páginas de 256 palabras
- 16 segmentos máximo por proceso

Actualmente en memoria hay un único proceso P que ocupa los marcos del 0 al 3, como muestra la figura (formato segmento/página):

Marcos	Seg. Pág
0	S0,4
1	S1,9
2	S0,5
3	S2,1

- Exponga el formato de las direcciones lógicas y físicas indicando su estructura, distribución de bits y tamaños

0.75 puntos

3

a)

Tamaño de página es de 256 palabras= 2^8 , el número de bits para desplazamiento es de 8 (tanto en marco como en página)
 Dado que hay 1024 marcos, se necesitan 10 bits para identificarlo.
 Por tanto las direcciones físicas son de 18 bits. 10 para marco y 8 para desplazamiento.

Las direcciones lógicas son de 18 bits de los cuales: 4 son para segmento (4 como máximo), 4 para página y 8 para desplazamiento.



- b) Utilizando un algoritmo ÓPTIMO de reemplazo de páginas con ámbito LOCAL muestre la evolución del contenido de la memoria para el conjunto de referencias:

0x1358 0x056D 0x1901 0x216E 0x2178 0x3BD8 0x0567 0x1345 0x1367

0x0590 0x0500 0x0D33 0x0D23 0x1340 0x3B34 0x3B21 0x21AB

1.0 puntos

3

b) Número total de Fallos de Página= 3 fallos con reemplazo

Mar		S1,3	S0,5	S1,9	S2,1	S3,B	S0,5	S1,3	S0,5	S0,D	S1,3	S3,B
0	S0,4	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3	S1,3
1	S1,9	S1,9	S1,9	S1,9	S1,9	S3,B	S3,B	S3,B	S3,B	S3,B	S3,B	S3,B
2	S0,5	S0,5	S0,5	S0,5	S0,5	S0,5	S0,5	S0,5	S0,5	S0,D	S0,D	S0,D
3	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1	S2,1

- c) Determine la dirección física que se generará al emitir la dirección lógica 0x21AB, si la ocupación de memoria fuese la siguiente

Marco	Seg, Pág
0	S0,4
1	S1,9
2	S0,5
3	S2,1

0.5 puntos

3 c)	<i>La dirección 0x21AB es la correspondiente al segmento 2, página 1 desplazamiento AB. Como la página del segmento ha sido ubicada en el marco 3 la dirección lógica en hexadecimal es la 0x003AB</i>
-----------------------	--

- d) Razone si para el mismo conjunto de referencias los algoritmos de reemplazo FIFO, LRU o 2ª oportunidad, producirían un mayor o menor número de fallos de página

0.25 puntos

3 d)	<i>El algoritmo óptimo se basa en elegir como víctima la página que más tiempo vamos a tardar en referenciar, y es por lo tanto el que consigue el menor número de fallos. Este algoritmo no se puede implementar en la realidad ya que no podemos predecir cuales serán las siguientes referencias.</i>
-----------------------	--



Dado el siguiente conjunto de referencias a páginas realizados por los procesos A, B y C (indicadas como proceso/nº de página):

A1,A2,A1,A3,B1,B2,B2,B2,B3,B4,B3,B5,C2,C3,C4,A1,C5,C6,C4,

C3,B5,B4,B3,A2,A3,A1,B4,B5, B3,C2,C3,C4,B2,B1,A5,A6,A7

a) Calcule el área activa en el instante final para un tamaño de ventana de 6

b) Asumiendo un modelo de área activa con tamaño de ventana 6, determine si, en el instante final, podría llegar o no a producirse hiperpaginación, en un sistema con 8 marcos de memoria principal

1.0 punto

4	<p>a)Área activa para un tamaño de ventana de 6</p> <p><i>Primero obtenemos las referencias de cada proceso</i></p> <p><i>A= A1,A2,A1,A3,A1,A2,A3,A1,A5,A6,A7</i></p> <p><i>B=B1,B2,B2,B2,B3,B4,B3,B5,B5,B4,B3 B4,B5, B3,B2,B1</i></p> <p><i>C= C2,C3,C4,C5,C6,C4,C3,C2,C3,C4</i></p> <p><i>Para A su ventana en el instante final es AA(A)= A2,A3,A1,A5,A6,A7.</i></p> <p><i>Tamaño AA(A)= 6</i></p> <p><i>Para B su ventana en el instante final es AA(B)= B3,B4,B5,B3,B2,B1.</i></p> <p><i>Tamaño AA(B)=5</i></p> <p><i>Para C su ventana en el instante final es AA(C)= C6,C2,C3,C4</i></p> <p><i>Tamaño AA(C)=4</i></p> <p>b) ¿puede producirse hiperpaginación?</p> <p><i>Por tanto 6+5+4=15 > nº de marcos. No puede albergarse todas las páginas del AA de los procesos en memoria</i></p> <p><i>El área activa de todos los procesos es mayor que el número de marcos → puede llegar a producirse hiperpaginación</i></p>
---	---

Considerando el mapa de memoria de un proceso UNIX, diga si las siguientes sentencias son verdaderas o falsas:

1.0 puntos

5	<table> <tr> <th data-bbox="260 1348 379 1382">V/F</th><th data-bbox="379 1348 1398 1382">Sentencias</th></tr> <tr> <td data-bbox="260 1382 379 1415">F</td><td data-bbox="379 1382 1398 1415">Todas las regiones del mapa de memoria tienen soporte en un archivo</td></tr> <tr> <td data-bbox="260 1415 379 1485">V</td><td data-bbox="379 1415 1398 1485">La región de código con soporte en el archivo ejecutable del proceso tiene permisos de lectura y de ejecución, pero no de escritura</td></tr> <tr> <td data-bbox="260 1485 379 1554">V</td><td data-bbox="379 1485 1398 1554">Para poder mapear un archivo en memoria el proceso ha de ejecutar previamente la llamada "open" sobre el archivo</td></tr> <tr> <td data-bbox="260 1554 379 1659">F</td><td data-bbox="379 1554 1398 1659">Al proyectar un proceso P en su mapa de memoria un fichero f con éxito obtiene un descriptor de fichero asociado a f, que permite a P leer/escribir el contenido de f utilizando las llamadas al sistema read() y write()</td></tr> <tr> <td data-bbox="260 1659 379 1729">V</td><td data-bbox="379 1659 1398 1729">Cuando un proceso realiza una llamada a exec() entonces en su mapa de memoria cambia la región de código que tendrá como soporte el nuevo archivo ejecutable.</td></tr> <tr> <td data-bbox="260 1729 379 1798">V</td><td data-bbox="379 1729 1398 1798">Cuando un proceso crea un proceso hijo con fork() el mapa de memoria del hijo justo después de fork() es idéntico al de su padre</td></tr> <tr> <td data-bbox="260 1798 379 1899">F</td><td data-bbox="379 1798 1398 1899">El fichero con el código ejecutable, de un programa que hace uso de funciones de biblioteca, ocupa un mayor espacio en disco, si se enlazan dinámicamente las bibliotecas</td></tr> </table>	V/F	Sentencias	F	Todas las regiones del mapa de memoria tienen soporte en un archivo	V	La región de código con soporte en el archivo ejecutable del proceso tiene permisos de lectura y de ejecución, pero no de escritura	V	Para poder mapear un archivo en memoria el proceso ha de ejecutar previamente la llamada "open" sobre el archivo	F	Al proyectar un proceso P en su mapa de memoria un fichero f con éxito obtiene un descriptor de fichero asociado a f, que permite a P leer/escribir el contenido de f utilizando las llamadas al sistema read() y write()	V	Cuando un proceso realiza una llamada a exec() entonces en su mapa de memoria cambia la región de código que tendrá como soporte el nuevo archivo ejecutable.	V	Cuando un proceso crea un proceso hijo con fork() el mapa de memoria del hijo justo después de fork() es idéntico al de su padre	F	El fichero con el código ejecutable, de un programa que hace uso de funciones de biblioteca, ocupa un mayor espacio en disco, si se enlazan dinámicamente las bibliotecas
V/F	Sentencias																
F	Todas las regiones del mapa de memoria tienen soporte en un archivo																
V	La región de código con soporte en el archivo ejecutable del proceso tiene permisos de lectura y de ejecución, pero no de escritura																
V	Para poder mapear un archivo en memoria el proceso ha de ejecutar previamente la llamada "open" sobre el archivo																
F	Al proyectar un proceso P en su mapa de memoria un fichero f con éxito obtiene un descriptor de fichero asociado a f, que permite a P leer/escribir el contenido de f utilizando las llamadas al sistema read() y write()																
V	Cuando un proceso realiza una llamada a exec() entonces en su mapa de memoria cambia la región de código que tendrá como soporte el nuevo archivo ejecutable.																
V	Cuando un proceso crea un proceso hijo con fork() el mapa de memoria del hijo justo después de fork() es idéntico al de su padre																
F	El fichero con el código ejecutable, de un programa que hace uso de funciones de biblioteca, ocupa un mayor espacio en disco, si se enlazan dinámicamente las bibliotecas																

Dado el siguiente fragmento de código en C y primitivas POSIX corresponde a un proceso que al ejecutarlo hereda de su padre una tabla de descriptores de archivos con los tres descriptores estándar

```
.....
int tubo[2];
int fd;

pipe(tubo);
/** Rellene tabla del proceso P1

if (!(pid=fork())) {
    dup2(tubo[1],1);
    close(tubo[0]);
    close(tubo[1]);
/**Rellene tabla del proceso P2
    execlp("/bin/cat", "cat", "fich1", NULL);
}

if(!(pid=fork())){
    dup2(tubo[0],0);
    fd=open("result",O_WRONLY |O_CREAT|O_TRUNC,0666);
    dup2(fd,1);
    close(tubo[0]);
    close(tubo[1]);
    close(fd);
/**Rellene tabla del proceso P3
    execlp("/usr/bin/wc", "wc", "-l",NULL);
}
close(tubo[0]);
close(tubo[1]);
while(pid != wait(&status));
}
```

- a) Rellene la tabla de descriptores de archivo para cada uno de los procesos que intervienen en los instantes marcados con el comentario “Rellene tabla...”
- b) Indique cuál sería la línea de ordenes (comandos, tubos, redirecciones, etc..) que se ejecutaría al lanzar la ejecución de este código.

1.0 puntos

6	a)																																									
	<div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> <p><i>Tabla de P1</i></p> <table border="1"> <tr><td>0</td><td>STDIN</td></tr> <tr><td>1</td><td>STDOUT</td></tr> <tr><td>2</td><td>STDERR</td></tr> <tr><td>3</td><td>tubo[0]</td></tr> <tr><td>4</td><td>tubo[1]</td></tr> <tr><td>5</td><td></td></tr> <tr><td>6</td><td></td></tr> </table> </div> <div style="text-align: center;"> <p><i>Tabla de P2</i></p> <table border="1"> <tr><td>0</td><td>STDIN</td></tr> <tr><td>1</td><td>tubo[1]</td></tr> <tr><td>2</td><td>STDERR</td></tr> <tr><td>3</td><td></td></tr> <tr><td>4</td><td></td></tr> <tr><td>5</td><td></td></tr> <tr><td>6</td><td></td></tr> </table> </div> <div style="text-align: center;"> <p><i>Tabla de P3</i></p> <table border="1"> <tr><td>0</td><td>tubo[0]</td></tr> <tr><td>1</td><td>result</td></tr> <tr><td>2</td><td>STDERR</td></tr> <tr><td>3</td><td></td></tr> <tr><td>4</td><td></td></tr> <tr><td>5</td><td></td></tr> <tr><td>6</td><td></td></tr> </table> </div> </div>	0	STDIN	1	STDOUT	2	STDERR	3	tubo[0]	4	tubo[1]	5		6		0	STDIN	1	tubo[1]	2	STDERR	3		4		5		6		0	tubo[0]	1	result	2	STDERR	3		4		5		6
0	STDIN																																									
1	STDOUT																																									
2	STDERR																																									
3	tubo[0]																																									
4	tubo[1]																																									
5																																										
6																																										
0	STDIN																																									
1	tubo[1]																																									
2	STDERR																																									
3																																										
4																																										
5																																										
6																																										
0	tubo[0]																																									
1	result																																									
2	STDERR																																									
3																																										
4																																										
5																																										
6																																										
	<p>b)</p> <pre>\$cat fich1 wc -l >result De forma genérica \$P2 fich1 P3 >result</pre>																																									

Una partición de 32MBytes, de los cuales 2Mbytes están ocupados con estructuras propias del sistema de archivos, está organizada en bloques de 512 bytes, el puntero a bloque es de 32 bits. Calcule:

- Tamaño máximo de un archivo si se utiliza asignación indexada simple, con un solo bloque de índices por archivo
- Tamaño máximo de un archivo si se utiliza asignación enlazada

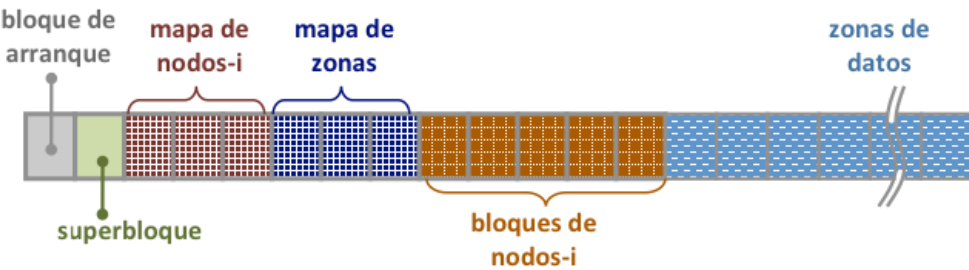
1.0 puntos

7	<p>a) Tamaño máximo de un archivo si se utiliza asignación indexada simple</p> <p><i>Los punteros a bloque son de 32 bits → 4 bytes</i> <i>Con punteros de 32 bits el número máximo de bloques que podemos direccionar son : $2^{32} = 4 \times 1024 \times 1024 \times 1024$ bloques, cualquier tamaño máximo de archivo debe ser igual o inferior a este número de bloques.</i> <i>Cada bloque de 512 bytes contiene $512/4 = 128$ punteros a bloque.</i> <i>Por tanto $128 \times 512 = 2^7 \times 2^9 = 2^{16} = 64$ KBytes</i></p> <p>b) Tamaño máximo con asignación enlazada</p> <p><i>El espacio que podemos considerar libre en el disco para crear el archivo es de 32MBytes-2 MBytes =30 MBytes</i> <i>Cada bloque es de 512bytes, por tanto el número total de bloques del disco es:</i> <i>$30 \text{ MBytes} / 512 \text{ Bytes} = 30 \times 1024 / 512 = 60 \times 1024$ bloques</i></p> <p><i>En cada bloque tendremos 512 bytes= 4 bytes para puntero al siguiente+508 bytes de información propia del archivo → Tamaño máximo para datos del archivo= $508 \times 60 \times 1024 = 30480$ KBytes</i></p>
---	--

Se ha formateado una partición de 16 Mbytes con Minix, utilizando los tamaños estándar y creando un nodo-i por cada 2kbytes Los tamaños estándar de Minix son los siguientes: nodos-i de 32bytes (7 punteros directos, 1 indirecto, 1 doble indirecto), entradas de directorios de 16bytes, 1zona=1bloque=1024bytes, puntero a zona de 16 bits.

Determine la estructura de la partición indicando cada uno de los elementos que la componen y el tamaño de cada una de ellas en bloques o zonas.

1.0 puntos

8	 <p><i>1 bloque de arranque</i> <i>1 bloque de superbloque</i> <i>Numero de nodos-i = $16 \text{ Mbyte} / 2 \text{ Kbytes} = 8 \text{ Kbyte} = 8192$ nodos-i</i> <i>Mapa de nodos-i >= $8192 \text{ bits} = 1 \text{ Kbyte} = 1$ bloque para Mapa nodos-i</i> <i>Numero de zonas = $16 \text{ Mbytes} / 1 \text{ Kbyte} = 16 \text{ K}$ zonas</i> <i>Mapa de zonas = $16384 \text{ bits} = 2 \text{ Kbyte} = 2$ bloques Mapa zonas</i> <i>Espacio para nodos-i = $8192 \text{ nodos-i} \times 32 \text{ bytes} = 256 \text{ Kbytes} = 256$ bloques</i> <i>Zona de datos = $16384 \text{ zonas} - (1 \text{ arranque} + 1 \text{ superbloque} + 1 \text{ mapa nodos} + 2 \text{ mapa zonas} + 256 \text{ de nodos-i}) = 16384 - 261 = 16123$ zonas para datos</i></p>
---	--