

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, cuya valoración se indica en cada una de ellas.
- Recuerde que debe justificar sus cálculos o respuestas para obtener buena calificación

1. Un computador gestiona su espacio lógico de memoria de 1 MB mediante paginación, con páginas de 512 Bytes.

El mapa de memoria de cierto proceso reserva espacio para tres regiones: una para 1100 instrucciones de 4 Bytes (todas contiguas), otra para 130 variables globales de 8 Bytes (también contiguas) y una tercera para la pila. El sistema asigna dinámicamente el mínimo número posible de páginas para la pila.

(1,2 puntos = 0,3 + 0,3 + 0,3 + 0,3)

1	<p>a) ¿Cuántos bits se reservan para número de página en la dirección lógica?</p> <p>Las páginas son de 2^9 bytes, quedarán $20-9=11$ bits para la página.</p>
	<p>b) ¿Cuántas páginas se necesitan para almacenar las instrucciones de este proceso? ¿y para sus variables globales?</p> <p>Hacen falta $1100 \times 4 = 4400$ Bytes para las instrucciones, que se alojan en $4400/512 = 9$ páginas. Las variables globales ocupan $130 \times 8 = 1040$ Bytes; $1040/512 = 3$ páginas</p>
	<p>c) En el momento en que la pila ha crecido hasta ocupar 1800 bytes, ¿cuántas entradas de la tabla de páginas serán válidas para el proceso?</p> <p>La pila necesitará 4 páginas. Por lo tanto, la tabla tendrá $4+9+3=16$ entradas válidas</p>
	<p>d) Calcule (en Bytes) la fragmentación interna en el mismo momento que en el apartado c)</p> <p>Si hay 16 páginas válidas, con 8192 bytes de capacidad conjunta, y el proceso requiere $4400+1040+1800 = 7240$ bytes, la fragmentación interna será de $8192-7240 = 952$ bytes</p>

2. Sea un sistema de tiempo compartido con planificador a corto plazo y una única cola de procesos preparados. En el instante $t=0$ llegan a su cola de preparados los procesos A, B y C en este orden. Se trata de tres procesos orientados a CPU con idéntico perfil de ejecución:

1000 CPU	10 E/S	1000 CPU	10 E/S	10 CPU
----------	--------	----------	--------	--------

Determine de forma justificada el tiempo de retorno y el tiempo espera de cada proceso (A, B y C) para:

(1,2 puntos = 0,5 + 0,5 + 0,2)

2

a) Un algoritmo de planificación FCFS.

FCFS: El primero que llega es el primero que se sirve. En primer lugar A ocupará la CPU y no la abandonará hasta que haya finalizado su ráfaga de CPU. Las ráfagas de E/S son muy pequeñas comparadas con respecto a las de CPU de cada proceso. Un sistema de tiempo compartido solapa las ráfagas de E/S con las de CPU de un proceso distinto. La ocupación de CPU será: $1000A+1000B+1000C+1000A+1000B+1000C+10A+10B+10C \rightarrow 100\%$ de ocupación. $Tret(A)=6010, Tret(B)=6020, Tret(C)=6030,$
 $Tesp(A)=6010-2030=3980, Tesp(B)=6020-2030=3990, Tesp(C)=6030-2030=4000.$

Prep- BC -CA(-10)-AB(-10)-BC(-10)-CA(-10)-AB(-10)- B - C -----
CPU- 1000A- 1000B - 1000C - 1000A - 1000B - 1000C - 10A- 10B- 10C
E/S->-----10A-----10B-----10C-----10A-----10B-----10C -----

Tiempo de retorno			Tiempo de Espera		
A=	B=	C=	A=	B=	C=

b) Un algoritmo de planificación RR con $q=10ut$.

RR con $q=10ut$: Los procesos ocupan la CPU de forma rotatoria cada $10ut$. Primero A ocupa la CPU por $10ut$, después B por $10ut$ y luego C y así sucesivamente. El quantum q coincide con las ráfagas de E/S($10ut$) que se solapan en el tiempo con las de CPU de un proceso distinto. En la CPU tendremos: $(10A+10B+10C)*100$ veces + $(10A+10B+10C)*100$ veces + $10A+10B+10C$
 La ocupación de CPU es del 100%. $Tret(A)=6010, Tret(B)=6020, Tret(C)=6030.$
 $Tesp(A)=6010-2030=3980, Tesp(B)=6020-2030=3990, Tesp(C)=6030-2030=4000.$

Prep-A(-1010)B(-1010)C(-1000)-A(-1010)B(-1010)C(-1010)- B - C ---
CPU--(10A + 10B + 10C)* 100 -(10A + 10B+ 10C)*100 -10A-10B-10C
E/S->-----10A-10B-----10C-----10A-10B-10C -----

Tiempo de retorno			Tiempo de Espera		
A=	B=	C=	A=	B=	C=

c) Asuma un coste para los cambios de contexto y justifique entre RR y FCFS cuál sería el algoritmo con mejor rendimiento para esta situación.

Con FCFS únicamente se dan unos 9 cambios de contexto, mientras que en el RR son unos 603. Considerando los cambios de contextos necesarios tendrá un mejor rendimiento FCFS.

Los cambios de contexto consumen tiempo, ya que debe intervenir en ellos la CPU y ejecutar código del SO. Este tiempo generaría un incremento del tiempo de retorno y de espera de cada proceso.

3. Dado el siguiente código cuyo fichero ejecutable ha sido generado con el nombre “Ejemplo1”.

```

1  /*** Ejemplo1.c ***/
2  #include "todas_las_cabeceras_necesarias.h"
3  #define N 2
4
5  main() {
6      int i=0, j=0;
7      pid_t pid;
8      while (i<N) {
9          pid = fork();
10         if (pid ==0) {
11             for (j=0; j<i ; j++) {
12                 fork();
13                 printf("Child i=%d, j=%d \n", i, j);
14                 exit(0);
15             }
16         } else {
17             printf("Parent i=%d, j=%d \n", i, j);
18             while (wait(NULL) != -1);
19         }
20         i++;
21     }
22     exit(0);
23 }

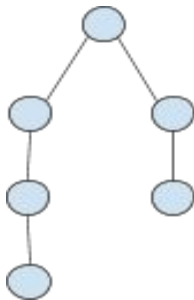
```

Indique de forma justificada:

(1,2 puntos = 0,6 + 0,6)

- 3 a) El número de procesos que se generan al ejecutarlo y dibuje el esquema de parentesco entre procesos.

Se generan 6 procesos con la siguiente relación entre ellos



- b) Indique los mensajes que se imprimirán por pantalla al ejecutar el código “Ejemplo1.c”

Soy padre con i=0,j=0
 Soy padre con i=1,j=0
 Soy padre con i=1,j=0
 Soy hijo con i=1,j=0
 Soy hijo con i=1,j=0
 Soy hijo con i=1,j=0
 Soy hijo con i=1,j=0

4. Dado el siguiente programa `hilos.c`, en el que la función `rever()` invierte la cadena que se le pasa como argumento:

1	<code>#include "Los_necesarios.h"</code>	17	<code>int main(int argc, char* argv[]){</code>
2	<code>#define ARG_MAX 20</code>	18	<code>int i;</code>
3	<code>int done=0;</code>	19	<code>pthread_t thr[ARG_MAX];</code>
4		20	<code>pthread_attr_t atrib;</code>
5	<code>void *rever(void *ptr){</code>	21	<code>pthread_attr_init(&atrib);</code>
6	<code>char temp;</code>	22	
7	<code>char *pfirst= (char*) ptr;</code>	23	<code>for(i=1;i<argc;i++){</code>
8	<code>char *plast= pfirst+strlen(pfirst)-1;</code>	24	<code>pthread_create(&thr[i],&atrib,</code>
9	<code>while (plast > pfirst){</code>		<code>rever, (void *)argv[i]);</code>
10	<code>temp= *pfirst;</code>	25	
11	<code>*pfirst= *plast; *plast= temp;</code>	26	<code>for(i=argc-1;i>0;i--){</code>
12	<code>pfirst++; plast--;</code>	27	<code>pthread_join(thr[i],NULL);</code>
13	<code>}</code>	28	<code>printf("%s ",argv[i]);</code>
14	<code>done++;</code>	29	<code>}</code>
15	<code>}</code>	30	<code>}</code>
16			

Conteste, justificando sus respuestas, suponiendo que se compila y ejecuta de la siguiente forma:

```
gcc hilos.c -o hilos -lpthread
./hilos once upon a time
```

(1,0 puntos = 0,25 + 0,25 + 0,25 + 0,25)

4	a) ¿Cuál es el número máximo de hilos que pueden llegar a ejecutarse concurrentemente?
	5 hilos. Se crea uno para cada uno de los 4 argumentos, más el hilo principal.
	b) ¿Qué se muestra en la salida estándar tras la ejecución del programa?
	“emit a nopus eno” Cada uno de los 4 hilos invierte concurrentemente uno de los argumentos. El hilo principal espera correctamente la inversión de cada argumento y los va imprimiendo también en orden invertido de argumentos, del último al primero.
	c) ¿De qué manera podría afectar a la salida generada por el programa la eliminación de la línea 27?
	El hilo principal imprimiría los argumentos sin esperar a que hubieran sido invertidos completamente. El orden de los argumentos estaría invertido, pero algunos de ellos posiblemente no.
	d) En el programa original, si se eliminara la línea 27 y se inserta en la línea 25 este código: <code>while (done < argc-1);</code>
	¿Funcionaría correctamente, obteniéndose la misma salida del programa original?
	Podría no funcionar correctamente, porque se produce una condición de carrera con la variable compartida <code>done</code> , ya que los hilos la modifican (incrementan) concurrentemente sin ningún mecanismo de sincronización. Esto hace que no pueda asegurarse que el valor final de esta variable sea finalmente igual al número de hilos <code>rever</code> que han finalizado, como se pretende.

5. Se desea gestionar el aforo de un parque de atracciones limitado a 5000 personas. Para ello se dispone de un proceso que implementa las tareas de entrada y salida del parque mediante dos tipos de hilos que realizan las funciones “func_entrar” y “func_salir” respectivamente. Estas dos funciones deben controlar que no se produzcan más entradas cuando el aforo está lleno y que no se contabilicen salidas cuando el recinto está vacío. Además se desea conocer en todo momento el aforo real del recinto, para lo cual se utilizará una variable llamada “aforo” que los hilos irán modificando adecuadamente.

Para resolver los problemas de sincronización que surgen se utilizan tres semáforos:

- mutex : controla el acceso a la sección crítica
- plazas_libres : gestiona el número de plazas libres en el recinto
- personas : gestiona el número de personas que han entrado en el recinto

- a) Complete el código del programa principal con la inicialización de los semáforos definidos
- b) Complete el código de las funciones “func_entrar” y “func_salir”, realizando las operaciones necesarias sobre los semáforos definidos.

(1 puntos = 0,5 + 0,5)

5	<p>a)</p> <pre> #include <semaphore.h> #define N 5000 int aforo; sem_t mutex, plazas_libres, personas; pthread_t entrar, salir; void main () { // Inicialización de semáforos sem_init(&mutex, 0, 1); sem_init(&plazas_libres, 0, N); sem_init(&personas, 0, 0); pthread_create(&entrar, NULL , func_entrar, NULL); pthread_create(&salir, NULL , func_salir, NULL); ... }</pre>		
	<p>b)</p> <table border="1" data-bbox="183 1512 1423 1982"> <tr> <td data-bbox="183 1512 805 1982"> <pre> void *func_entrar(void *p) { while (1) { sem_wait(&plazas_libres) sem_wait(&mutex) aforo=aforo+1; sem_post(&mutex) sem_post(&personas) } }</pre> </td><td data-bbox="805 1512 1423 1982"> <pre> void *fun_salir(void *p) { while (1) { sem_wait(&personas) sem_wait(&mutex) aforo=aforo-1; sem_post(&mutex) sem_post(&plazas_libres) } }</pre> </td></tr> </table>	<pre> void *func_entrar(void *p) { while (1) { sem_wait(&plazas_libres) sem_wait(&mutex) aforo=aforo+1; sem_post(&mutex) sem_post(&personas) } }</pre>	<pre> void *fun_salir(void *p) { while (1) { sem_wait(&personas) sem_wait(&mutex) aforo=aforo-1; sem_post(&mutex) sem_post(&plazas_libres) } }</pre>
<pre> void *func_entrar(void *p) { while (1) { sem_wait(&plazas_libres) sem_wait(&mutex) aforo=aforo+1; sem_post(&mutex) sem_post(&personas) } }</pre>	<pre> void *fun_salir(void *p) { while (1) { sem_wait(&personas) sem_wait(&mutex) aforo=aforo-1; sem_post(&mutex) sem_post(&plazas_libres) } }</pre>		

6. Sea un sistema con **memoria virtual**, **paginación por demanda** y algoritmo de **reemplazo LRU GLOBAL**. Las páginas son de 4KB, la memoria principal es de 40 KB (10 marcos) y el tamaño máximo de un proceso de 16KB (4 páginas). El algoritmo LRU se implementa mediante contadores, anotando el instante de tiempo en que se referencia la página. La asignación de marcos en memoria principal es en orden creciente. En el instante $t=10$, existen 3 procesos cuyo contenido de sus tablas de páginas:

Tabla de páginas del proceso A				Tabla de páginas del proceso B				Tabla de páginas del proceso C			
Pág.	Marco	Contador	Bit Val.	Pág.	Marco	Contador	Bit Val.	Pág.	Marco	Contador	Bit.Val.
0	2	4	v	0	0	2	v	0	4	10	v
1	5	1	v	1	3	3	v	1	8	7	v
2			i	2			i	2	9	8	v
3	1	5	v	3			i	3			i

Considere el estado del sistema en $t = 10$ y conteste a las siguientes preguntas:

- Indique el proceso y número de página que ocupa cada marco de la memoria principal. Rellene para ello la tabla propuesta con el siguiente formato: A0 corresponde a la página 0 del proceso A.
- A partir de $t=10$, se referencian las siguientes direcciones lógicas (**direcciones en hexa**): (A, 2F2C), (C, 3001), (A, 11C1), (B, 3152). Rellene el contenido de la memoria principal y las tablas de páginas al finalizar dichas referencias..

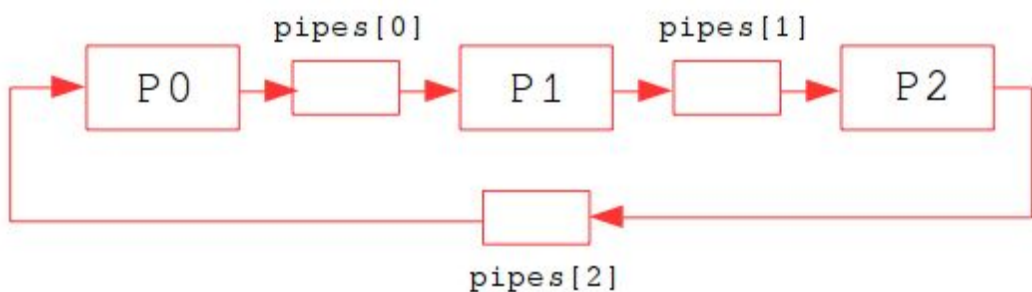
(1,2 puntos = 0,6 + 0,6)

6	a)	<div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px;"> <p>(TABLA apartado a))</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th>Nº marco</th> <th></th> </tr> <tr><td>0</td><td>B0</td></tr> <tr><td>1</td><td>A3</td></tr> <tr><td>2</td><td>A0</td></tr> <tr><td>3</td><td>B1</td></tr> <tr><td>4</td><td>C0</td></tr> <tr><td>5</td><td>A1</td></tr> <tr><td>6</td><td>--</td></tr> <tr><td>7</td><td>--</td></tr> <tr><td>8</td><td>C1</td></tr> <tr><td>9</td><td>C2</td></tr> </table> </div> <div style="border: 1px solid black; padding: 5px;"> <p>(TABLA apartado b))</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th>Nº marco</th> <th></th> </tr> <tr><td>0</td><td>B3</td></tr> <tr><td>1</td><td>A3</td></tr> <tr><td>2</td><td>A0</td></tr> <tr><td>3</td><td>B1</td></tr> <tr><td>4</td><td>C0</td></tr> <tr><td>5</td><td>A1</td></tr> <tr><td>6</td><td>A2</td></tr> <tr><td>7</td><td>C3</td></tr> <tr><td>8</td><td>C1</td></tr> <tr><td>9</td><td>C2</td></tr> </table> </div> </div>	Nº marco		0	B0	1	A3	2	A0	3	B1	4	C0	5	A1	6	--	7	--	8	C1	9	C2	Nº marco		0	B3	1	A3	2	A0	3	B1	4	C0	5	A1	6	A2	7	C3	8	C1	9	C2																													
Nº marco																																																																											
0	B0																																																																										
1	A3																																																																										
2	A0																																																																										
3	B1																																																																										
4	C0																																																																										
5	A1																																																																										
6	--																																																																										
7	--																																																																										
8	C1																																																																										
9	C2																																																																										
Nº marco																																																																											
0	B3																																																																										
1	A3																																																																										
2	A0																																																																										
3	B1																																																																										
4	C0																																																																										
5	A1																																																																										
6	A2																																																																										
7	C3																																																																										
8	C1																																																																										
9	C2																																																																										
TABLAS DE PÁGINAS DE LOS PROCESOS A, B y C (apartado b)																																																																											
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th colspan="4">Tabla de páginas de A</th> <th colspan="4">Tabla de páginas de B</th> <th colspan="4">Tabla de páginas de C</th> </tr> <tr> <th>Pág</th><th>Mar.</th><th>Cont.</th><th>Bit Val.</th> <th>Pág</th><th>Mar.</th><th>Cont.</th><th>Bit Val.</th> <th>Pág</th><th>Mar.</th><th>Cont.</th><th>Bit Val.</th> </tr> <tr> <td>0</td><td>2</td><td>4</td><td>v</td> <td>0</td><td></td><td></td><td>i</td> <td>0</td><td>4</td><td>10</td><td>v</td> </tr> <tr> <td>1</td><td>5</td><td>13</td><td>v</td> <td>1</td><td>3</td><td>3</td><td>v</td> <td>1</td><td>8</td><td>7</td><td>v</td> </tr> <tr> <td>2</td><td>6</td><td>11</td><td>v</td> <td>2</td><td></td><td></td><td>i</td> <td>2</td><td>9</td><td>8</td><td>v</td> </tr> <tr> <td>3</td><td>1</td><td>5</td><td>v</td> <td>3</td><td>0</td><td>14</td><td>v</td> <td>3</td><td>7</td><td>12</td><td>v</td> </tr> </table>				Tabla de páginas de A				Tabla de páginas de B				Tabla de páginas de C				Pág	Mar.	Cont.	Bit Val.	Pág	Mar.	Cont.	Bit Val.	Pág	Mar.	Cont.	Bit Val.	0	2	4	v	0			i	0	4	10	v	1	5	13	v	1	3	3	v	1	8	7	v	2	6	11	v	2			i	2	9	8	v	3	1	5	v	3	0	14	v	3	7	12	v
Tabla de páginas de A				Tabla de páginas de B				Tabla de páginas de C																																																																			
Pág	Mar.	Cont.	Bit Val.	Pág	Mar.	Cont.	Bit Val.	Pág	Mar.	Cont.	Bit Val.																																																																
0	2	4	v	0			i	0	4	10	v																																																																
1	5	13	v	1	3	3	v	1	8	7	v																																																																
2	6	11	v	2			i	2	9	8	v																																																																
3	1	5	v	3	0	14	v	3	7	12	v																																																																

7. Suponga que se ejecuta sin errores el siguiente código C, que crea 3 procesos que se comunican mediante 3 tubos, conteste de forma razonada a los apartados siguientes:

<pre> 1 /** pipes.c */ 2 #include "todas_cabeceras_necesarias.h" 3 #define nproc 3 4 #define nt 2 5 int main(int argc, char* argv[]) { 6 int pipes[nproc][2]; 7 int i, j, pid, i_1; 8 9 for (i=0; i<nproc; i++) 10 pipe(pipes[i]); 11 for (i=0; i<nproc; i++) { 12 pid = fork(); 13 if (pid == 0) { 14 dup2(pipes[i][1], STDOUT_FILENO); 15 i_1 = i_anterior(i, nproc); 16 dup2(pipes[i_1][0], STDIN_FILENO); 17 for (j=0; j<nproc; j++) { 18 close(pipes[j][0]); 19 close(pipes[j][1]); 20 } 21 comp_and_com(i, nt); 22 exit(0); 23 } 24 } 25 for (i=0; i<nproc; i++) { 26 close(pipes[i][0]); 27 close(pipes[i][1]); 28 } 29 while (wait(NULL) != -1); 30 exit(0); 31 } </pre>	<pre> 31 int comp_and_com(id, nturns) { 32 int v, n; 33 n = 0; 34 while (n < nturns) { 35 if (id == 0 && n == 0) { 36 v = 0; 37 n++; 38 write(STDOUT_FILENO, &v, sizeof(v)); 39 } else { 40 read(STDIN_FILENO, &v, sizeof(v)); 41 v = v + 1; 42 n++; 43 write(STDOUT_FILENO, &v, sizeof(v)); 44 } 45 } 46 if (id == 0) { 47 read(STDIN_FILENO, &v, sizeof(v)); 48 fprintf(stderr, "V FINAL VALUE: %d\n", 49 v); 50 } 51 return 0; 52 } 53 54 int i_anterior(int i, int np) { 55 if (i > 0) return (i-1) 56 else if (i == 0) return (np-1); 57 } </pre>
---	--

(1,0 puntos = 0,6 + 0,4)

7	<p>a) Dibuje el esquema de comunicación que se genera entre los procesos creados con fork()</p>  <pre> graph LR P0[P0] -- pipes[0] --> P1[P1] P1 -- pipes[1] --> P2[P2] P2 -- pipes[2] --> P0 </pre>
	<p>b) ¿Explique que imprimirá la sentencia fprintf de la línea 48?</p> <p>fprintf en la línea 48 imprime "5" ya que P0 envía a P1 el valor 0, P1 envía a P2 el valor 1 y P2 envía a P0 el valor 2. Como nt es 2 entonces se produce una 2da vuelta, P0 envía 3 a P1, P1 envía 4 a P2 y P2 envía 5 a P0 que finalmente lo imprime.</p>

--	--

8. Dado el siguiente listado del contenido de un directorio en un sistema POSIX:

```
drwxr-xr-x  2 pepe    alumno      4096 ene  8   2013  .
drwxr-xr-x 11 pepe    alumno      4096 ene 10   14:39  ..
-rwsr-xr-x  1 pepe    alumno    1139706 ene  9   2013  copia
-rwxr-sr--  1 pepe    alumno    1139706 ene  9   2013  anyade
-rw-----  1 pepe    alumno      634310 ene  9   2013  f1
-rw----rw-  1 pepe    alumno      104157 ene  9   2013  f2
-rw-rw----  1 pepe    alumno      634310 ene  9   2013  f3
```

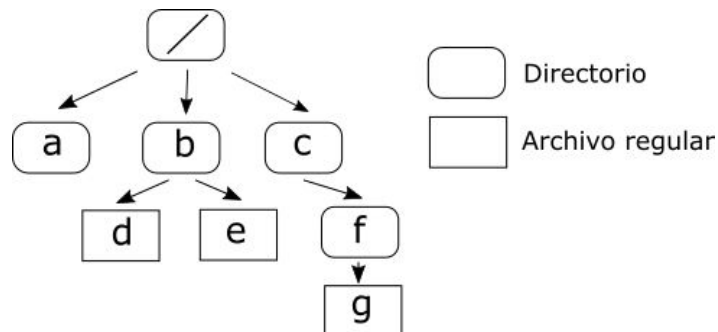
Donde los programas `anyade` y `copia` son programas que requieren el nombre de dos archivos como argumentos. El programa `añade` al final del archivo pasado como segundo argumento el contenido del primero, mientras que `copia` simplemente sustituye el contenido del archivo del segundo argumento por el del primero. Rellene la tabla e indique en caso de éxito cuales son los permisos que se comprueban y, en caso de error, cuál es el permiso que falla y porqué.

(1,0 puntos = 0,25 + 0,25 + 0,25 + 0,25)

8				
	Usuario	Grupo	Orden	¿Funciona?
	ana	profes	./anyade f1 f2	NO
	Justifique Para ejecutar la orden hace falta permisos de ejecución sobre anyade, de lectura sobre f1 y de escritura sobre f2. Falla al no tener permisos de ejecución sobre anyade, ya que al no ser el propietario ni del grupo hay que comparar la tercera tripleta y en ésta no hay permiso de ejecución			
	ana	profes	./copia f2 f4	SI
	Justifique Para ejecutar la orden hace falta permisos de ejecución sobre copia, de lectura sobre f2 y de escritura sobre el directorio (.) ya que f4 no existe. (ana, profes) puede ejecutar copia y pasa a ser (pepe, profes), al cambiar el usuario efectivo al del propietario de los ficheros puede tanto leer f2 como escribir en el directorio (".") para crear f4			
	pau	alumno	./anyade f2 f3	NO
	Justifique Para ejecutar la orden hace falta permisos de ejecución sobre anyade, de lectura sobre f2 y de escritura sobre f3. Falla al intentar leer f2, ya que los del grupo alumno no tienen permisos de lectura sobre el fichero			
	pepe	alumno	./copia f3 f1	SI
	Justifique Para ejecutar la orden hace falta permisos de ejecución sobre copia, de lectura sobre f3 y de escritura sobre f1. El propietario (primera tripleta) tiene todos los permisos para ejecutar la orden y tiene permisos de lectura y escritura tanto en f3 como en f1			

--	--

9. En un sistema de archivos minix se contiene el siguiente árbol de archivos:



(1,2 puntos = 0,5 + 0,7)

- 9 a) Indique el contenido de cada directorio de este sistema de archivos, suponiendo que los i-nodos ocupados por cada elemento son: a(4), b(5), c(8), d(12), e(22), f(10) y g(16)

/		a		b		c		f	
1	.	4	.	5	.	8	.	10	.
1	..	1	..	1	..	1	..	8	..
4	a			12	d	10	f	16	g
5	b			22	e				
8	c								

- b) Considerando zonas de 1 KByte, punteros a zona de 32 bits, e i-nodos con 7 punteros directos, 1 puntero simple indirecto y 1 puntero doble indirecto, obtenga el número de zonas ocupadas por el archivo *d* si su tamaño es de 2 MBytes.

2 MBytes / 1 KByte = 2048 zonas ----> 2048 punteros a zona

Con los 7 punteros directos se direccionan 7 zonas = 7 KBytes

Con el simple indirecto se utiliza una zona que contiene N° punteros por zona
= 1024 Bytes / 4 Bytes = 256 punteros

Así pues se tienen: 7 KB + 256 KB = 263 KB, quedan por direccionar:

2048 KB - 263 KB = 1785 KB que tendrán que direccionarse con el puntero doble indirecto, necesitando:

1785 KB / 256 = 6,97 -> 7 zonas para punteros de 2do nivel.

El archivo ocupa pues: 2048 zonas para contenido de datos + (1 zona de simple indirecto que contiene 256 punteros a zona + 1 zona del doble indirecto que contiene 7 punteros a zona + 7 zonas que contienen un total de 1785 punteros a zona) zonas de metadatos con punteros = 2057 zonas