

Nombre y apellidos alumno:

DNI:

Examen final de teoría de SO

Justifica todas tus respuestas del examen. Las respuestas no justificadas se considerarán erróneas.

Gestión de memoria(2puntos)

Analiza el siguiente programa (mem.c) y responde a las siguientes preguntas. El programa mem.c se ejecuta sobre un sistema Linux, con la optimización Copy On Write, el tamaño de página es de 4KB, el tamaño de un entero (int) es de 4 bytes, el tamaño de las direcciones de memoria es de 32 bits, y el código del programa cabe en 1 página.

```
1. #define pagesize 4096
2. int array10[100 * pagesize];
3. int i;
4. int main ()
5. {
6.     int array20[100 * pagesize];
7.     int *array30;
8.     for (i = 0; i < 100 * pagesize; i++) array10[i] = 10;
9.
10.    array30 = malloc (100 * sizeof(int) * pagesize);
11.
12.    if (fork () == 0)
13.    {
14.        for (i = 0; i < 100 * pagesize; i++) array20[i] = 20;
15.    }
16.    else
17.        for (i = 0; i < 100 * pagesize; i++) array30[i] = 30;
18.
19.    free(array30);
20. }
```

- a) **(0,5 puntos)** Indica para las variables *array10*, *array20* y *array30* en qué región de memoria del proceso están localizadas y dónde está localizado su contenido.

Nombre y apellidos alumno:

DNI:

b) **(0,5 puntos)** Justifica cuántos frames de región de pila se están consumiendo en la línea 17

c) **(0,5 puntos)** Supón que paramos la ejecución en la línea 17. Consultamos las herramientas del sistema y vemos que el tamaño del heap para el proceso padre es de 533 páginas. Justifica este tamaño del heap.

d) **(0,5 puntos)** Modifica las líneas de código que consideres para que la reserva y liberación de memoria dinámica sea mediante llamadas a sistema

Nombre y apellidos alumno:

DNI:

Procesos y signals (3 puntos)

Nos dan el siguiente código (programa n_steps) que genera una jerarquía de procesos.

```

1. uint sigusr1_received = 0;
2. void notifica(char *pid_str)
3. {
4.     int pid;
5.     pid = atoi(pid_str);
6.     if (pid) kill(pid, SIGUSR1);
7. }
8. void f_sigusr1(int s)
9. {
10. sigusr1_received = 1;
11.}
12.void espera()
13.{
14. alarm(5);
15. while(sigusr1_received == 0);
16. alarm(0);
17.}
18.void do_work(int step){// Ejecuta cálculo }
19.void espera_hijos()
20.{
21. char buffer[256];
22. int hijos = 0;
23. while(waitpid(-1,NULL, WNOHANG)> 0) hijos++;
24. sprintf(buffer,"Hijos terminados %d\n", hijos);
25. write(1, buffer, strlen(buffer));
26.}

```

```

27.void main(int argc, char *argv[])
28.{
29. int ret , step;
30. char buffer[256], buffer1[256];
31. step = atoi(argv[1]);
32.
33. if (step > 0 ){
34.     ret = fork();
35.     if (ret > 0){
36.         sprintf(buffer,"%d", ret);
37.         sprintf(buffer1,"%d", step-1);
38.         execlp(argv[0], argv[0], buffer1, buffer, NULL);
39.     }
40. }else{
41.     do_work(step);
42.     notifica(argv[2]);
43.     espera_hijos();
44.     exit(0);
45. }
46. espera();
47. do_work(step);
48. notifica(argv[2]);
49. exit(0);
50.}

```

El objetivo del código es que cada proceso (excepto uno) espere la recepción del signal SIGUSR, luego ejecute la función do_work y notifique al siguiente que ya puede ejecutarse. Hay uno que no espera sino que ejecuta la función e inicia la cadena. Conociendo esta funcionalidad básica, y sabiendo que el código está incompleto, contesta las siguientes preguntas. Se ha eliminado el control de errores por simplicidad. Sabemos, además, que al inicio del main los signals están bloqueados y las acciones asociadas son las acciones por defecto.

- a) **(0,5 puntos)** Dibuja la jerarquía de procesos que genera si ejecutamos el programa de la siguiente forma “n_steps 5 0”. Asigna un número a cada proceso para referirte a ellos e indica que parámetros recibiría cada proceso. Si el proceso muta muestra el último código y argumentos.

- b) **(0,25 puntos)** La función “espera_hijos”, ¿Realiza una espera activa o bloqueante? ¿Podemos saber qué valor se imprimiría para la variable “hijos”?

Nombre y apellidos alumno:

DNI:

- c) **(0,25 puntos)** La función “espera_hijos”, ¿Es correcto el flag que utiliza si queremos que haga una espera bloqueante y la variable “hijos” tenga el número de hijos creados por cada proceso? Si no es correcto indica que valor deberíamos usar en el flag.

- d) **(0,25 puntos)** Indica qué signals recibirá cada proceso y qué procesos ejecutarán la función do_work

- e) **(0,5 puntos)** Tal y como está el código, ¿qué deberíamos añadir y donde (línea de código) para garantizar que los procesos puedan recibir los señales que se usan y tratar el SIGUSR1 mediante la función f_sigusr1 ? (asume este cambio en los siguientes apartados).

- f) **(0,5 puntos)** ¿Qué modificaciones harías en la función “espera” para que no consuma tiempo de CPU? (Conservando el control de los 5 segundos tal cual está ahora)

- g) **(0,5 puntos)** ¿Qué modificación harías en la función “espera_hijos” para detectar, para cada proceso, si ha ejecutado o no la función do_work?

- h) **(0,25 puntos)** ¿Se cumplen los requisitos para poder ejecutar la llamada a sistema kill de la línea 6?

Nombre y apellidos alumno:

DNI:

Procesos y pipes (3 puntos)

La figura 1 muestra el código de los programas `fusion_encryptada` y `encryptar` (se omite el código de control de errores).

```

1. /* fusion_encryptada */
2. #define N 10
3. main(int argc, char *argv[]){
4.     int i,ret,fd_pipe[N][2];
5.     char c;
6.     i=0; ret=1;
7.     while ((i<argc-1) && (ret>0)){
8.         pipe(fd_pipe[i]);
9.         ret=fork();
10.        if (ret>0) i++;
11.    }
12.    if (ret == 0) {
13.        close(0); open(argv[i+1], O_RDONLY);
14.        dup2(fd_pipe[i][1],1);
15.        execlp("./encryptar","encryptar",(char *)0);
16.    }
17.    for (i=0;i<argc-1;i++){
18.        close(fd_pipe[i][1]);
19.    }
20.    for (i=0;i<argc-1;i++){
21.        while((ret=read(fd_pipe[i][0],&c,sizeof(char))>0){
22.            write(1,&c,sizeof(char));
23.        }
24.    }
25.    exit(0);
26. }

```

```

27. /* encryptar */
28. char crypt (char *c) {
29.     /* Codigo para encriptar el carácter c
30.      * No es relevante para el ejercicio
31.      */
32. }
33. main(int argc, char *argv[]){
34.     char c1,c2;
35.     while ((ret=read(0, &c1, sizeof(char))>0) {
36.         c2=crypt(c1);
37.         write(1, &c2, sizeof(char));
38.     }
39.     exit(0);
40. }

```

figura 1 Código del programa `fusion_encryptada` y `encryptar`

Desde el directorio en el que tenemos los dos códigos ejecutamos el siguiente comando: `./fusion_encryptada f1 f2 > f3`. Donde `f1`, `f2`, y `f3` son ficheros que existen y su contenido es respectivamente: "Feliz", "año", y "nuevo".

Suponiendo que ninguna llamada a sistema devuelve error, responde razonadamente a las siguientes preguntas

- a) **(0,5 puntos)** Dibuja la jerarquía de procesos que se creará como consecuencia de este comando. Representa también la(s) pipe(s) creada(s), indicando de qué pipe(s) lee y/o escribe cada proceso. Añade un identificador a cada proceso y pipe para poder referirte a ellos en los siguientes apartados.

- b) **(0,5 puntos)** Completa la figura con el estado de las tablas, considerando a todos los procesos de la jerarquía y suponiendo que todos se encuentran entre la línea 11 y la línea 12. Añade las filas y tablas que necesites.

Nombre y apellidos alumno:

DNI:

Tabla de Canales		Tabla de Ficheros abiertos				Tabla de iNodo	
Entrada TFA		refs	modo	Pos. l/e	Entrada T.inodo	refs	inodo
0		0	rw	-	0	0	1 tty1
1		1				1	
2		2				2	
3		3				3	

- c) **(0,5 puntos)** Completa la figura con el estado de las tablas, suponiendo que el proceso inicial va a ejecutar el exit. Considera los procesos de la jerarquía que estén vivos en ese momento. Añade las filas y tablas que necesites.

Tabla de Canales		Tabla de Ficheros abiertos				Tabla de iNodo	
Entrada TFA		refs	modo	Pos. l/e	Entrada T.inodo	refs	inodo
0		0	rw	-	0	0	1 tty1
1		1				1	
2		2				2	
3		3				3	

- d) **(0,5 puntos)** Rellena la siguiente tabla indicando qué procesos de la jerarquía leen y/o escriben en los dispositivos.

	Lectores	Escritores	Justificación
<i>f1</i>			
<i>f2</i>			
<i>f3</i>			
<i>terminal</i>			

- e) **(0,5 puntos)** ¿Cuál será el contenido de f3 después de ejecutar este comando? ¿Se puede garantizar que si repetimos varias veces el mismo comando el contenido de f3 será siempre el mismo?

- f) **(0,5 puntos)** ¿Qué procesos acabarán la ejecución?


Nombre y apellidos alumno:

DNI:

Sistema de ficheros (2 puntos)

Tenemos un sistema de ficheros tipo Unix, con 12 punteros directos a bloques de datos y tres punteros indirectos (simple, doble y triple direcciones a bloque). El tamaño del puntero (a bloque, a inodo y el de la Tabla de Ficheros Abiertos) es de 4 Bytes. El tamaño de los bloques es de 4096 Bytes. Los tipos de ficheros son: directory, regular, fifo y softlink. Las tablas de Inodos y de Bloques de Datos se muestran en la figura 2. Ejecutamos un ls con las opciones long, inode (primera columna), all y Recursive (para ver el subdirectorio Datei):

```
murphy@numenor: /home/murphy/Examen$ ls -liaR
total 56
800170 drwxr-xr-x 3 murphy students 4096 Dec 23 11:47 .
787670 drwxr-xr-x 3 murphy students 4096 Dec 18 14:03 ..
800210 -rwxr-xr-x 1 murphy students 16920 Dec 23 11:47 caps
800177 drwxr-xr-x 2 murphy students 4096 Dec 23 11:06 Datei
800178 lrwxrwxrwx 1 murphy students 5 Dec 18 11:46 Folder -> Datei
800179 -rw-r--r-- 3 murphy students 4211 Dec 18 11:49 Lorem.bp
800179 -rw-r--r-- 3 murphy students 4211 Dec 18 11:49 text.txt
800176 prw-r--r-- 2 murphy students 0 Dec 18 11:45 tube
./Datei:
total 16
800177 drwxr-xr-x 2 murphy students 4096 Dec 23 11:06 .
800170 drwxr-xr-x 3 murphy students 4096 Dec 23 11:47 ..
800179 -rw-r--r-- 3 murphy students 4211 Dec 18 11:49 Ipsum.txt
800176 prw-r--r-- 2 murphy students 0 Dec 18 11:45 mypipe
```

- a) (0,5 puntos) Rellenad las celdas marcadas con  de las tablas de la figura 2 que tenéis al final del ejercicio.
- b) (0,25 puntos) En este sistema de ficheros, ¿cuál es el tamaño máximo de un fichero? ¿Por qué?

- c) (1,25 puntos) Ejecutamos sin errores, desde el terminal, el programa de la figura 2.
- a. (0,5 pts.) Al bucle (lin. 6-9), ¿cuántas llamadas a la `syscall` read se han hecho? ¿Cuántos bloques de datos se han leído?

- b. (0,5 pts.) Escribe los números de inodos y di cuántos bloques de datos se han visitado al ejecutar la línea 4?

- c. (0,25 pts.) ¿Qué estructuras de datos del kernel se han modificado al ejecutar la línea 5? ¿Qué inodos y cuántos bloques de datos se liberan?

Nombre y apellidos alumno:

DNI:

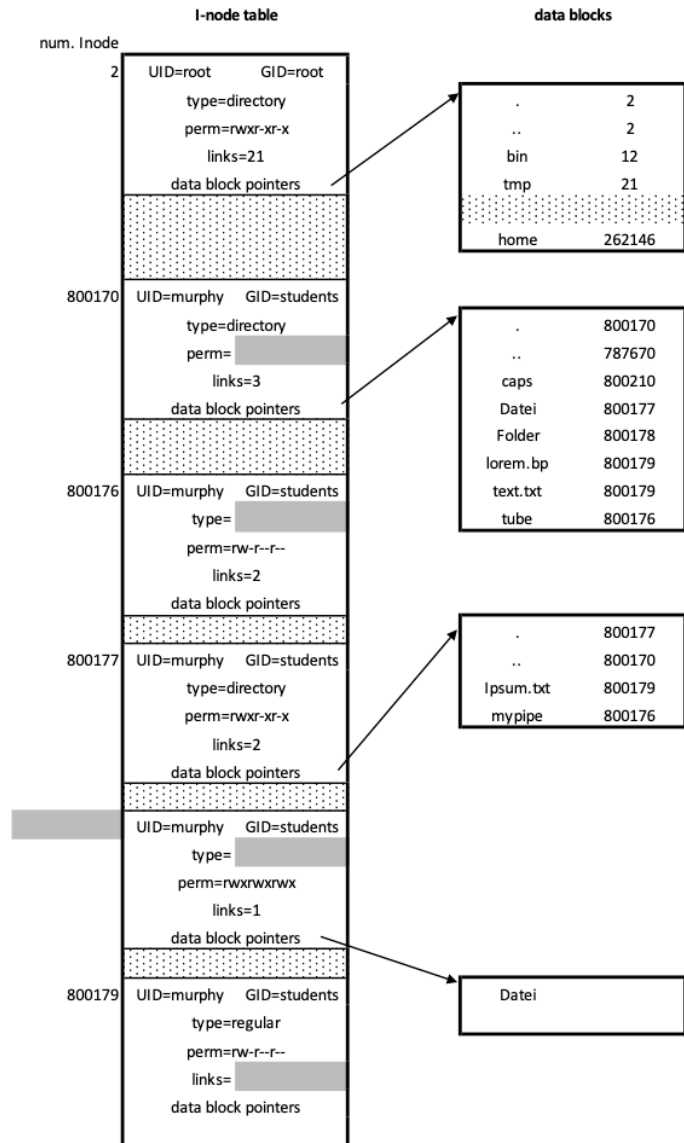


figura 2 Tabla de i-nodos y bloques

```

1. int main(int argc, char**argv) {
2.   char c;
3.   int ifd, ofd;
4.   ifd =
5.     open("/home/murphy/Examen/Folder/Ipsum.txt", O_RDONLY);
6.   ofd =
7.     open("new_IPSUM.TXT", O_WRONLY|O_CREAT|O_TRUNC, 0644);
8.   while (read(ifd, &c, 1) > 0) {
9.     if (c > 96 && c < 123) c = c - 32;
10.    write(ofd, &c, 1);
11.  }
12.  close(ifd); close(ofd);

```

figura 3 Código del programa caps.c