Nombre alumno:
DNI:

Examen final de teoría de SO

Justifica todas tus respuestas de este examen. Cualquier respuesta sin justificar se considerará errónea.

Preguntas Cortas (*2 puntos*)

1. *(0,5 puntos)* Define quantum (en (en el ámbito de una política de planificación Round Robin)

Es el tiempo máximo que un proceso puede estar en la CPU (estado RUN) de forma continua (por ráfaga)

2. (*0,5 puntos*) Indica llamadas a sistema que puedan bloquear signals en un proceso.

Sigaction, sigsuspend, sigprocmask

3. (*0,5 puntos*) ¿En qué caso el código de la llamada malloc(..) no invoca a ninguna syscall de reserva memoria?

Cuando en el heap actual del proceso hay suficiente espacio libre contiguo para cubrir la demanda de memòria de malloc.

4. (*0,5 puntos*) Indica en qué casos un sistema ejecutará código de kernel.

- Cuando tenga que atender a una interrupcion
- Cuando tanga que resolver una excepción
- Cuando el usuario invoque a una llamada a sistema

Nombre alumno:
DNI:

Nombre alumno:
DNI:

Procesos y pipes (*4 puntos*)

La Figura 1 contiene el código del programa *EjercicioProcs*. Omitimos el control de errores para facilitar la legibilidad del código.

```
1. sigalrm_trat(int n) {
2.         write(2, "Final!\n", 7);
3.         exit(0);
4. }
5. sigpipe_trat(int n) {
6.         write(2, "Final!\n", 7);
7.         while(waitpid(-1, NULL,0) > 0);
8.         exit(0);
9. }
10.  void leer_datos() {
11.  char c;
12.  char buf[80];
13.  sigset_t mask;
14.  sigfillset(&mask);
15.  sigdelset(&mask, SIGALRM);
16.  sigprocmask(SIG_SETMASK,&mask,0);
17.
18.  alarm(1);
19.  while(read(0, &c, sizeof(c)) > 0) {
20.           alarm(0);
21.           write(1, &c, sizeof(c));
22.           alarm(1);
23.  }
24.  alarm(0);
25.  }
26.
27.  main(int argc, char *argv[]) {
28.  struct sigaction sa;
29.  int fd, pipe_fd[2];
30.  int pidh=1, i=0;
31.  char c;
32.  int nprocs = atoi(argv[1]);
33.  sigset_t mask;
34.
35.  sigemptyset(&mask);
36.  sigaddset(&mask, SIGALRM);
37.  sigprocmask(SIG_BLOCK, &mask, 0);
38.  sa.sa_flags=0;
39.  sa.sa_handler=sigalrm_trat;
40.  sigfillset(&sa.sa_mask);
41.  sigaction(SIGALRM, &sa, NULL);
42.
43.  sa.sa_handler=sigpipe_trat;
44.  sigaction(SIGPIPE, &sa, NULL);
45.
46.  close(1);
47.  fd=open(argv[2],  O_WRONLY|O_CREAT|O_TRUNC, 0600);
```

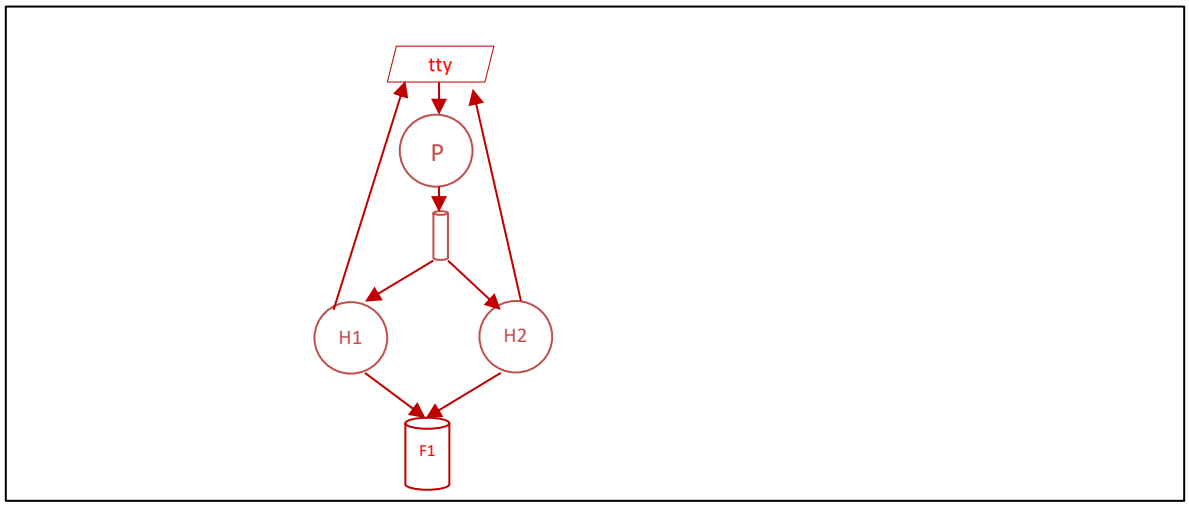
```
48.  pipe(pipe_fd);
49.  while ((i<nprocs) && (pidh>0)) {
50.           pidh=fork();
51.           i++;
52.  }
53.
54.  if (pidh == 0) {
55.           dup2(pipe_fd[0], 0);
56.           close(pipe_fd[0]); close(pipe_fd[1]);
57.           leer_datos();
58.           write(2,"Fin datos\n", 10);
59.           exit(0);
60.  }
61.  close(pipe_fd[0]);
62.  while(read(0, &c, sizeof(c)) > 0)
63.           write(pipe_fd[1], &c, sizeof(c));
64.
65.  close(pipe_fd[1]);
66.  while(waitpid(-1,NULL,0) > 0);
67.  write(2, "Fin de ejecucion\n",13);
68.
69.  }
```

Figura 1 Código de EjercicioProcs

Ponemos en ejecución este programa con el siguiente comando: ./*EjercicioProcs* 2 f1

1. *(0,5 puntos)* Representa la jerarquía de procesos que crea este código. Asigna a cada proceso un identificador para poder referirte a él en los siguientes apartados. Representa también a qué fichero, dispositivo o pipe accede cada proceso, indicando claramente si son accesos de lectura o de escritura

Nombre alumno:
DNI:



2. *(0,5 puntos)* Completa la siguiente figura con el estado de las tablas de canales, tabla de ficheros abiertos y tabla de inodos, suponiendo que todos los procesos están en la línea 53.

Tabla de Canales

	Entrada TFA
0	0
1	1
2	0
3	2
4	3
5	

Tabla de Ficheros abiertos

	refs	modo	Posición l/e	Entrada T.inodo
0	6	RW	---	0
1	3	W	0	1
2	3	R	---	2
3	3	W	---	2
4				
5				
6				

Tabla de iNodo

	refs	inodo
0	1	I-tty
1	1	I-f1
2	2	I-pipe
3		
4		
5		

TC. H1

0	0
1	1
2	0
3	2
4	3
5	

TC. H2

0	0
1	1
2	0
3	2
4	3
5	

3. Suponed que por teclado introducen los caracteres “abcdefg” y se pulsa ctrl-D (para indicar final de datos), todo ello en menos de 1 segundo. Contesta razonadamente a las siguientes preguntas.
   a) *(0,25 puntos)* ¿Cuál será el contenido del fichero f1 al final de la ejecución?

F1 contendrá las letras abcdefg pero puede ser que aparezcan en otro orden porque tenemos dos procesos leyendo de la misma pipe y escribiendo en el mismo fichero (compartiendo el puntero) sin ningún tipo de sincronización.

Nombre alumno:
DNI:

b) *(0,25 puntos)* Si ejecutamos varias veces el programa y con la misma entrada por teclado, ¿podemos garantizar que siempre será el mismo contenido?

No porque no hay ninguna sincronización. Puede ser que un proceso lea un carácter y antes de escribirlo le quiten el procesador para dárselo a otro, que entonces consiga leer y escribir y así se altera el orden

c) *(0,25 puntos)* ¿Qué mensajes veremos por pantalla?

Veremos los mensajes "Fin datos" de los hijos y el mensaje "Fin de ejecución" del padre. Las escrituras en el canal 1 van al fichero f1, el único canal asociado al terminal es el 2.

d) *(0,25 puntos)* Acabarán todos los procesos la ejecución?

Todos acaban la ejecución. El padre sale de su bucle de lectura cuando pulsan ctrl-D, entonces cierra el único extremo de escritura de la pipe y eso hace que los hijos salgan de sus bucles de lectura y acaben. Al acabar los dos hijos, el padre saldrá del bucle de waitpid y acabará.

e) *(0,5 puntos)* Si movemos la línea 47 a la posición 53, ¿afectará de alguna manera al contenido de f1? Si la respuesta es sí, indica cómo; si la respuesta es no justifica por qué.

En esa posición, el canal asociado al fichero ya no será el 1, que habrá sido ocupado por el extremo de lectura de la pipe. Por lo tanto el write en canal 1 que hacen los hijos devolverá error. Además, aunque se solucionara el tema del canal, los 3 procesos harían open del fichero y por lo tanto no compartirán el puntero de lect/escrit: las escrituras de un hijo sobreescribirán las del otro. Además, como se usa el flag TRUNC, un proceso podría borrar todo lo que el otro hubiera ya escrito.

4. Sobre la versión original del código. Suponed que por teclado introducen los caracteres "abc" en menos de 1 segundo, a continuación se esperan 3 segundos, y por último se introduce "d" (sin pulsar después ctrl-D). Contesta razonadamente a las siguientes preguntas.

a) *(0,25 puntos)* ¿Cuál será el contenido del fichero f1 al final de la ejecución?

En este caso, después de leer "abc" saltaría la alarma de los dos lectores y acabarían la ejecución. Así que el contenido será esas 3 letras (quizás en otro orden)

Nombre alumno:
DNI:

b) *(0,5 puntos)* ¿Qué signals recibirá cada proceso? ¿Podemos saber en qué línea de código estarán los procesos cuando lo reciben? Si la respuesta es si indica la línea. En cualquier caso, justifica tu respuesta.

Los dos hijos recibirán sigalrm ya que después de leer la c pasarán más de 1 segundo esperando a recibir otro carácter y saltará su temporizador. Entonces mostrarán un mensaje en pantalla y acabarán la ejecución. Al acabar la ejecución, el padre se quedará sin lectores y cuando escriba la "d" recibirá un SIGPIPE, mostrará el mensaje de final y acabará también la ejecución.

c) *(0,25 puntos)* ¿Acabarán todos los procesos la ejecución?

Si (tal y como se ha explicado en el apartado anterior)

d) *(0,25 puntos)* Si eliminamos la línea 61, ¿afectará en algo a los signals recibidos o a los procesos que acaban la ejecución?

Al eliminar esa línea, el padre consta como posible lector de la pipe por lo que cuando escriba el carácter d en la pipe no recibirá el SIGPIPE. Se quedará a la espera de que introduzcan un nuevo carácter y no acabará.

e) *(0,25 puntos)* Sobre la versión original del código, movemos todo el código de la función leer_datos a un programa separado llamado prog_lector en el mismo directorio, y substituimos la línea 57 por la siguiente línea de código:

```
57: execlp("./prog_lector","prog_lector",NULL);
```

¿Qué signals tratará cada proceso? ¿Qué mensajes veremos en pantalla?

En este caso los signals serán los mismos pero los hijos pierden la reprogramación del sigalrm y morirán sin escribir nada. Los mensajes que veremos son "Final" del padre.

Nombre alumno:
DNI:

Memoria (2 puntos, 0,5 por apartado)

Analiza detenidamente el código de la figura 2 (programa *mem*) y responde a las preguntas justificadamente. Considera que se ejecuta en un sistema Linux, con Copy on Write y el tamaño de página de 4KB. El tamaño del ejecutable *miprog* es el mismo que el de *mem*.

```
1.  /************** mem.c ****************/
2.  #define MIDAPAGINA 4096
3.  int matrix[MIDAPAGINA*16];
4.  void fgolafre(int j) {
5.  int i;
6.  for (i=j;i<MIDAPAGINA*16;i++) {
            matrix[i]=23;
7.  }
8.  }
9.  int main() {
10. int *vector;
11. vector=(int*)malloc(MIDAPAGINA*8);
12. if (fork()==0) {
            execlp("miprog","miprog",NULL);
13. }
14. else if (fork()==0) {
            vector[4]=73;
            exit(0);
15. }
16. fgolafre(MIDAPAGINA*4);
17. wait(NULL);
18. wait(NULL);
19. free(vector);
20. return 0;
21. }
```

figura 2 Código del programa mem

1. ¿En qué región de memoria está el contenido de la variable matrix? ¿Y el contenido de vector?

`matrix` es una variable global no inicializada, por tanto, estará en la zona de datos.

`vector` es una variable local, por tanto, estará en la pila, pero su contenido estará en el heap.

2. ¿Qué regiones de memoria se modificarán al ejecutarse la llamada a la función fgolafre?

El código de la función recorre parte de `matrix`, que está en la zona de datos. Pero también tiene parámetros y variables locales que están en la pila.

3. Justo después del malloc, parte del contenido del fichero maps del proceso padre es el siguiente :

Nombre alumno:
DNI:

```
/proc/29390$ cat maps
55a6e75e9000-55a6e75ea000 r-xp 00000000 08:01 1074816    /home/alumne/final/mem
55a6e77e9000-55a6e77ea000 r--p 00000000 08:01 1074816    /home/alumne/final/mem
55a6e77ea000-55a6e77eb000 rw-p 00001000 08:01 1074816    /home/alumne/final/mem
55a6e9248000-55a6e9269000 rw-p 00000000 00:00 0          [heap]
7fff35c97000-7fff35cb8000 rw-p 00000000 00:00 0          [stack]
```

4. ¿Cuántas páginas mide la región del heap? ¿Cuánto medía antes del malloc?

55a6e9269000 - 55a6e9248000 = 21000, es decir, 21 páginas (1000 en hexa es 4096 en base 10) . Muchas más de las 8 pedidas. Antes de esa línea el heap no existía.

5. ¿Qué proceso provoca más fallos de página debido al uso de COW? Si no hubiera COW, ¿qué proceso sería el más perjudicado?

El proceso padre escribe en más páginas compartidas, lo que provocará fallos de página para pedir nuevas páginas para el proceso hijo y copiarlas.

Si no hubiera COW, el primer proceso hijo sería el más perjudicado en cuanto a tiempo de creación en el fork. Habría que pedir memoria y copiar todas las páginas del padre, para inmediatamente desecharlas al cambiar la imagen del proceso en la llamada a execlp.

Nombre alumno:
DNI:

Sistema de ficheros (*2 puntos*)

Disponemos de un Sistema de Ficheros basado en I-nodos del cual sabemos lo siguiente:

- El superbloque ya está cargado en memoria.
- El tamaño de bloque es 1Kbyte (tanto para bloque de datos como para Inodo).
- El sistema dispone de Buffer Cache, que alberga tanto Inodos como bloques de datos, de suficiente tamaño como para que no llegue a activarse la política de reemplazo de su contenido.
- El fichero "file.dat" y "Hfile.dat" son hard-links al mismo fichero que contiene números enteros (en formato interno).
- El fichero "Sfile.dat" es un soft-link que contiene la ruta absoluta del fichero "file.dat".
- Ejecutamos las líneas de comando, con los correspondientes resultados, que muestra la
- figura 3 (NOTA: los símbolos con interrogante son valores que preguntaremos posteriormente).
- El ejecutable "prog.exe" tiene el código de alto nivel de la figura 4 y lo ejecutamos mediante la siguiente línea de comandos:

#> ./prog.exe /home/alumne/Hfile.dat /home/alumne/Sfile.dat

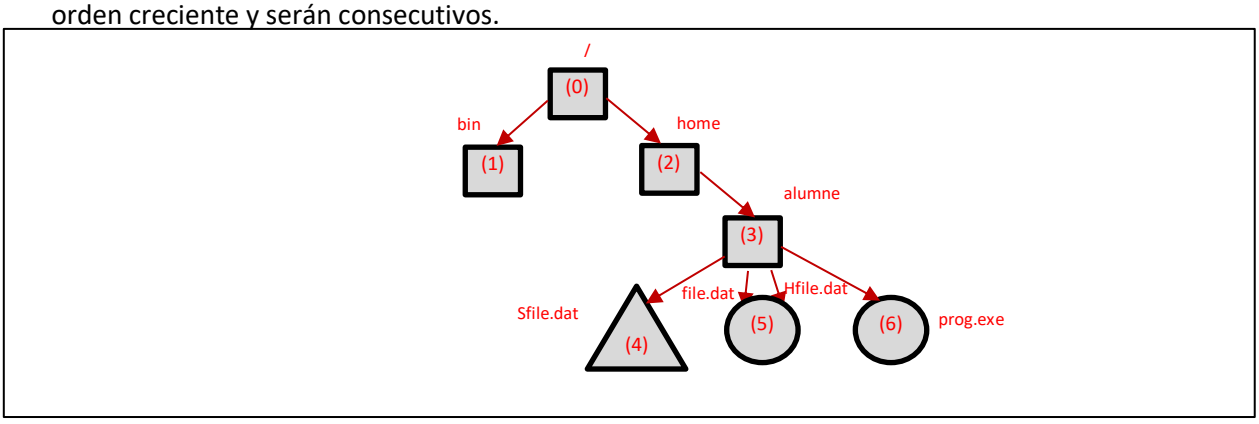
figura 3 Resultado comandos

```
#>pwd
/

#>ls –la
drwx------   ??? profe ac    1024 Dec 28 2018 .
drwx------   ??? profe ac    1024 Dec 28 2018 ..
drwx------   ??? profe ac    1024 Dec 28 2018 bin
drwx------   ??? profe ac    1024 Dec 28 2018 home

#>cd /home/alumne

#>ls –la
drwx------   ??? profe ac    1024 Dec 28 2018 .
drwx------   ??? profe ac    1024 Dec 28 2018 ..
-rwx------   ??? profe ac    2780 Dec 28 2018 prog.exe
-rwx------   ??? profe ac     600 Dec 28 2018 file.dat
-rwx------   ??? profe ac     600 Dec 28 2018 Hfile.dat
lrwx------   ??? profe ac      21 Dec 28 2018 Sfile.dat
```

figura 4 Código de prog.exe

```
1.  void main (int argc, char **argv){
2.    int fdi, fdo, num;
3.    fdi=open(argv[1], O_RDONLY);
4.    fdo=open(argv[2],O_WRONLY|O_CREAT, 0664);
5.    ret = read(fdi, &num, sizeof(int));
6.    lseek(fdo, -1 * sizeof(int), SEEK_END);
7.    write(fdo, &num, ret);
8.    close(fdi);
9.    close(fdo);
10. }
```

1. *(0,5 puntos)* Representa el grafo del Sistema de Ficheros que se puede conocer a partir de los datos indicados. Los cuadrados representan directorios, círculos para ficheros regulares y triángulos para soft-links. En cada nodo del grafo indica el número de inodo que representa. Supón que el ID de Inodo del directorio raíz es el 0 y que el ID de cada Inodo se asignará en orden creciente y serán consecutivos.



Nombre alumno:
DNI:

2. *(0,5 puntos)* Rellena la siguiente tabla de inodos y bloques de datos (BDs) con la información correspondiente a este sistema de ficheros. Supón que el id de Bloque de Datos se asignarán en orden creciente y serán consecutivos.

Listado de Inodos											
ID Inodo	0	1	2	3	4	5	6	7	8	9	10
#enlaces	4	2	3	2	1	2	1				
Tipo	dir	dir	dir	dir	link	dat	dat				
BDs	0	1	2	3	4	5	6, 7, 8				

Listado de BDs									
ID BD	0	1	2	3	4	5	6	7	8
Datos	. 0 .. 0 bin 1 home 2	. 1 .. 0	. 2 .. 0 alumne 3	. 3 .. 2 Sfile.dat 4 file.dat 5 Hfile.dat 5 prog.exe 6	/home /alumne /file.dat	integers	binario	binario	binario

3. *(0,5 puntos)* Indica, **razonando brevemente tu respuesta**, el número de accesos a disco que se realiza al ejecutar las llamadas a sistema de las siguientes líneas de código, así como indicar qué estructuras de datos de SO (vinculadas a E/S y SF) se ven modificadas por ellas.

Línea 3: I0 + B0 + I2 + B2 + I3 + B3 + I5
Nueva entrada en TC, TFO y T-Inodos

Línea 4: I0 + B0 + I2 + B2 + I3 + B3 + I4 + B4 + I0 + B0 + I2 + B2 + I3 + B3 + I5
Nueva entrada en TC, TFO y actualizar número de referencias de la T-Inodos

Línea 5: B5
Actualizar offset de la entrada correspondiente de la TFO

Línea 6: ---
Actualizar offset de la entrada correspondiente de la TFO

4. *(0,5 puntos)* Si borramos el fichero "file.dat", razona brevemente qué consecuencias tendrá sobre los ficheros "Hfile.dat" y "Sfile.dat".

El fichero "Hfile.dat" seguirá funcionando, porque se trata de un hard-link al mismo fichero.

El fichero "Sfile.dat" dejará de funcionar, porque es un soft-link a una ruta que ya no existe.