

Nombre alumno:

DNI:

Examen final de teoría de SO

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

Preguntas Cortas (2 puntos)

a) **(0,5 puntos)** Ejecutamos el siguiente comando:

```
$> ln A B
```

Sabiendo que “A” existe, “B” no existe, que disponemos permisos para crear elementos en el directorio actual y que hay espacio libre en el disco, el comando falla. ¿A qué puede ser debido?

b) **(0,5 puntos)** Define el concepto de fragmentación externa de memoria, indicando en qué modelo de gestión de memoria se puede producir.

Nombre alumno:

DNI:

c) **(0,5 puntos)** En un programa se ejecuta la siguiente línea:

```
ret = read(fd, buffer, 100);
```

Asumiendo que **fd** apunta a un dispositivo virtual válido y que **buffer** está bien declarado y tiene tamaño suficiente, indica al menos un caso en que tras finalizar **read** la variable **ret** tenga el siguiente intervalo de valores: $1 \leq \text{ret} \leq 99$

d) **(0,5 puntos)** Supón que conoces la dirección de entrada de la rutina de kernel que implementa el servicio de la llamada a sistema write. ¿Se podría hacer un *call* directamente a esa rutina desde una aplicación de usuario, asumiendo que pasamos correctamente los parámetros?

Nombre alumno:

DNI:

Gestión de memoria (1 Punto)

Tenemos el siguiente código del programa "memoria". Este programa se ejecuta en un sistema Linux con tamaño de página 4096 bytes y nuestro sistema implementa la optimización de COW.

```
1. #define M_SIZE 4096
2. void print_limit()
3. {
4.     char buffer[256];
5.     void* limit;
6.     limit = sbrk(0);
7.     sprintf(buffer, "Limite %p\n", limit);
8.     write(1, buffer, strlen(buffer));
9. }
10. void main(int argc, char *argv[])
11. {
12.     int ret, i, *p1;
13.     print_limit();
14.     p1 = sbrk(M_SIZE * sizeof(int));
15.     print_limit();
16.     for (i = 0; i < M_SIZE; i++) p1[i] = 0;
17.     ret = fork();
18.     /* A */
19.     sbrk(-1 * M_SIZE * sizeof(int));
20.     print_limit();
21. }
```

Al ejecutarlo tenemos la siguiente salida:

```
[user@login]$ ./memoria
Limite 0x1971000
Limite 0x1975000
Limite XXXXXXXXX
Limite YYYYYYYYY
```

Contesta a las siguientes preguntas:

a) Rellena las líneas punteadas con la respuesta correcta.

1. La variable p1 está en la región de memoria, la variable limit está en la región de memoria
2. El valor que veremos en las XXXXXXXX es y en las YYYYYYYY es

b) Si nos dan la siguiente información: el tamaño de la pila es 4096 bytes, el código de este programa ocupa 1000 bytes, la variable p1 ocupa 8 bytes y el tamaño de un int son 4 bytes:

1. ¿Cuántos marcos de página (páginas físicas) tendrán reservados para las regiones de pila en el punto A incluyendo los dos procesos?

Nombre alumno:

DNI:

2. ¿Cuántos marcos de página (páginas físicas) tendrán reservados para las regiones de heap en el punto A incluyendo los dos procesos?

Nombre alumno:

DNI:

Procesos y Signals (3 Puntos)

Analiza el código que te mostramos a continuación y responde a las preguntas de la manera más detallada posible dentro del espacio de que dispones. Supón que ejecutamos el programa desde el terminal con la siguiente línea de comandos: \$./a.out 2

- a) **(0,5 puntos)** Dibuja la jerarquía de procesos creada. Etiquétalos para poder referirte a ellos durante el resto de tu respuesta.

```
1. int beep = 0;
2. void ras(int s) {
3.     sigset_t nores, m;
4.     sigfillset(&nores);
5.     if (s == SIGALRM) {
6.         write(1, " FINAL!\n", 8);
7.         beep++;
8.     }
9.     if (beep < 1) {
10.        sigprocmask(SIG_SETMASK, &nores, &m);
11.        sigdelset(&m, SIGALRM);
12.        sigsuspend(&m);
13.    }
14.}
15.

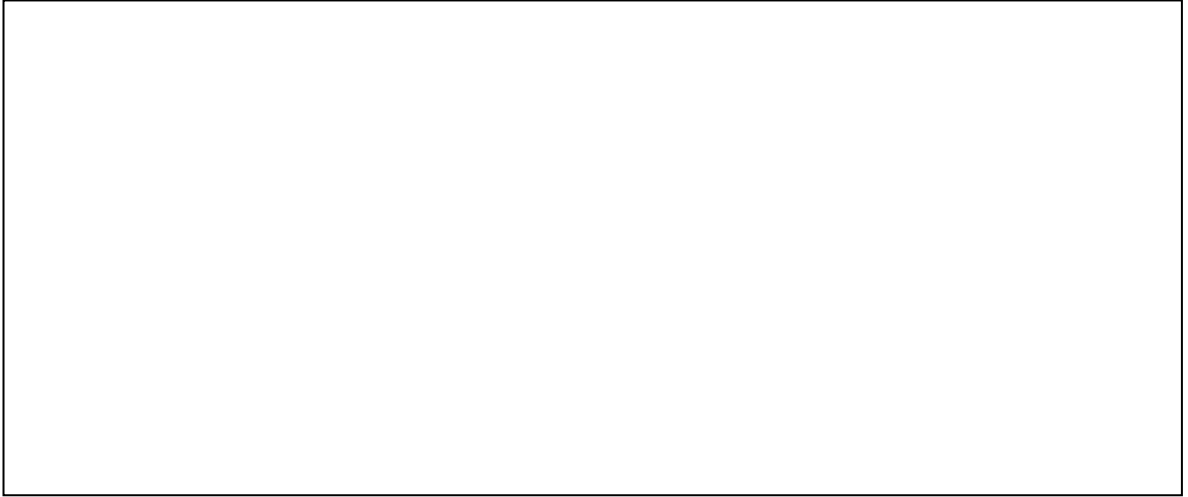
16.int main(int argc, char *argv[]) {
17.    int i;
18.    struct sigaction sa;
19.    int n = atoi(argv[1]);
20.    sa.sa_handler = ras;
21.    sigfillset(&sa.sa_mask);
22.    sa.sa_flags = SA_RESTART;
23.    for (i = 0; i < 32; i++) {
24.        sigaction(i, &sa, NULL);
25.        alarm(i);
26.    }
27.    for (i = 0; i < n; i++) fork();
28.    while (waitpid(-1, NULL, 0) > 0);
29.    write(1, "\tEXAMEN!", 8);
30.    exit(0);
31.}
```

- b) **(0,5 puntos)** ¿Qué signals están bloqueados cuando comenzamos a ejecutar la función ras()?

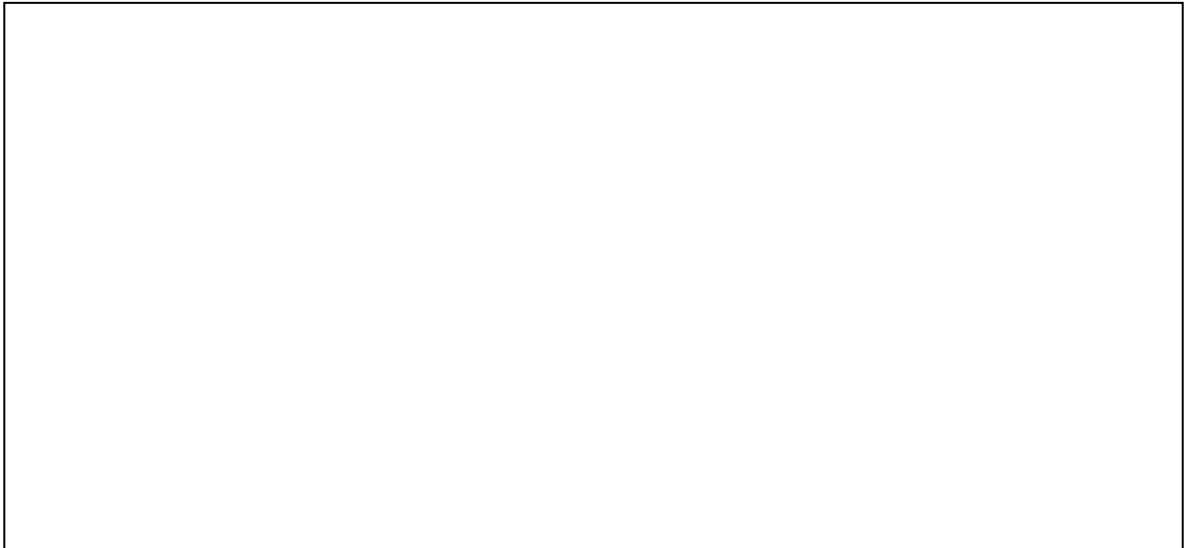
Nombre alumno:

DNI:

c) **(1,5 puntos)** ¿Qué procesos escriben "EXAMEN!"? ¿Y "FINAL!"?



d) **(0,5 puntos)**. ¿Qué línea de comandos tendrías que ejecutar para que todos los procesos llegasen a la línea 29, escribiendo por pantalla?

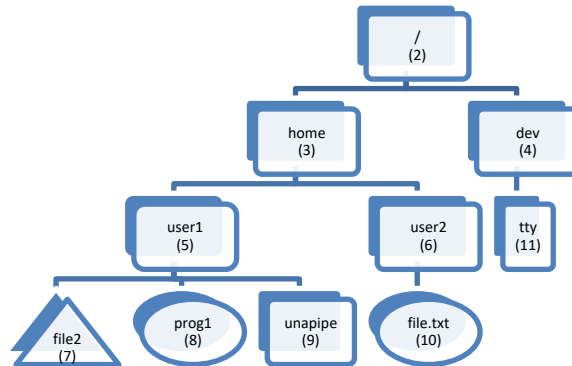


Nombre alumno:

DNI:

Entrada/Salida (4 Puntos)

Tenemos el siguiente SF basado en I-Nodos, con un tamaño de bloque de 1KB.



El fichero “file.txt” es un fichero de caracteres que contiene 2KB de datos. El fichero “file2” es un soft-link que apunta /home/user2/file.txt mediante un path absoluto. El fichero “prog1” es un ejecutable que ocupa menos de un bloque y es el resultado de compilar el siguiente código fuente:

```

1. main() {
2.   char c;
3.   int ret, fd;
4.   ret = fork();
5.   if (ret == 0) {
6.     fd=open("/home/user1/unapipe", O_WRONLY);
7.     while (read(0, &c, sizeof(c)) > 0)
8.       write(fd, &c, sizeof(c));
9.     exit(0);
10.  }
11. fd=open("/home/user1/unapipe", O_RDONLY);
12. while(read(fd, &c, sizeof(c)) > 0)
13.   write(1, &c, sizeof(c));
14. waitpid(-1, NULL, 0);
15. }
  
```

- a) **(0,75 puntos)** Completa las siguientes tablas con la información que falta para representar esta jerarquía. El campo “path” sólo se utiliza para los soft links y contiene el path del fichero referenciado.

| ID Inodo | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|-----------------------|---|---|---|---|---|----------------------|---|---|-----|----|----|
| #enlaces | | | | | | 1 | 1 | 1 | 1 | 1 | |
| Tipo | d | d | d | d | d | l | - | p | - | c | |
| Path | - | - | - | - | - | /home/user2/file.txt | - | - | - | - | |
| Tabla de índices a BD | 0 | 1 | 2 | 3 | 4 | | 5 | | 6,7 | | |

| ID BD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-----------|------|-------|---|---|---|--------|-------|-------|---|---|
| Contenido | . | . | | | | codigo | datos | datos | | |
| | .. | .. | | | | | | | | |
| | home | user1 | | | | | | | | |
| | dev | user2 | | | | | | | | |

Nombre alumno:

DNI:

b) **(3,25 puntos)** Supón que el directorio actual de trabajo es /home/user1 y que ejecutamos el siguiente comando: `./prog1 < /home/user1/file2 > /home/user1/file3.txt`

1. **(0,5 puntos)** Completa el estado de las siguientes estructuras de datos suponiendo que los procesos se encuentran justo después del fork (entre la línea 4 y 5).

| Tabla de Canales | | Tabla de Ficheros abiertos | | | | Tabla de iNodo | |
|------------------|--|----------------------------|------|-----------------|--------------------|----------------|----------|
| Entrada TFA | | refs | modo | Posición l/e | Entrada T.inodo | refs | inodo |
| 0 | | 0 | | | | 0 | 1 l tty1 |
| 1 | | 1 | | | | 1 | |
| 2 | | 2 | | | | 2 | |
| 3 | | 3 | | | | 3 | |
| 4 | | 4 | | | | 4 | |

2. **(0,5 puntos)** Describe brevemente lo que hace este comando. ¿Acabará la ejecución? Justifica tu respuesta

3. **(0,5 puntos)**. Indica cómo quedarán las estructuras de datos del apartado a) cuando el bucle de la línea 11 haya completado 2048 iteraciones. Si necesitas utilizar nuevos inodos o nuevos bloques asígnalos secuencialmente.

| ID Inodo | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|-----------------------|---|---|---|---|---|----------------------|---|---|-----|----|----|
| #enlaces | | | | | | 1 | 1 | 1 | 1 | 1 | |
| Tipo | d | d | d | d | d | l | - | p | - | c | |
| Path | - | - | - | - | - | /home/user2/file.txt | - | - | - | - | |
| Tabla de índices a BD | 0 | 1 | 2 | 3 | 4 | | 5 | | 6,7 | | |

| ID BD | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-----------|----------------------------|---------------------------|---|---|---|------------|-------|-------|---|---|
| Contenido | . 2 .. 2 home dev | . .. user1 user2 | | | | codig o | datos | datos | | |

Justificación:

