

Nombre alumno:

DNI:

Primer parcial de teoría de SO

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

Preguntas cortas (2,5 puntos)

- a) **(0,5 puntos)** Explica cómo o qué mecanismo se utiliza para invocar a rutinas de la librería de sistema.

- b) **(0,5 puntos)** Enumera qué estructuras de gestión de signals tiene un proceso y para cada una de ellas cómo cambiarían si el proceso muta.

- c) **(0,5 puntos)** Supón una política de planificación de procesos no apropiativa. Indica que circunstancias hacen que un proceso abandone el estado RUN.

- d) **(0,5 puntos)** ¿Es posible que la rutina de servicio de una interrupción provoque que un proceso abandone la CPU? Justifícalo con un ejemplo.

Nombre alumno:

DNI:

- e) **(0,5 puntos)** En un SO multiprogramado, ¿es posible que un proceso se quede bloqueado en un waitpid para siempre?

Nombre alumno:

DNI:

Gestión de procesos (4 puntos)

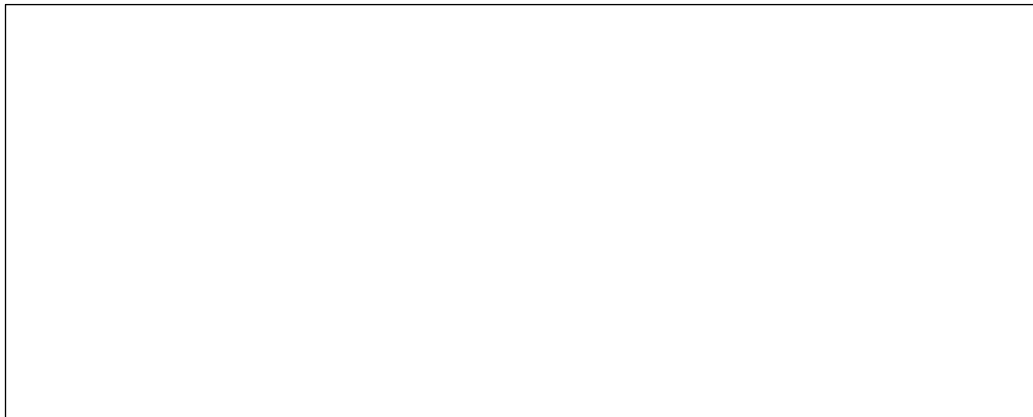
La figura 1 muestra el código del programa arbol.c (se omite el control de errores para facilitar la legibilidad del código). Desde el directorio donde se encuentra el programa, lo ponemos en ejecución con el siguiente comando: `./arbol 3`

```
1. /* arbol */
2.
3. main(int argc, char *argv[]){
4.     char buf[80];
5.     int st;
6.     int ret = 0;
7.     int nprocs = 0;
8.     int niveles = atoi(argv[1]);
9.     if (niveles > 0) {
10.         ret = fork();
11.         if (ret > 0) {
12.             ret = fork();
13.         }
14.         if (ret == 0) {
15.             niveles--;
16.             sprintf(buf,"%d",niveles);
17.             execlp("./arbol", "arbol", buf, (char *)0);
18.         }
19.     }
20.     if (ret > 0) {
21.         while ((ret=waitpid(-1, &st, 0))>0) {
22.             nprocs=nprocs+1;
23.             if (WIFEXITED(st)) {
24.                 nprocs+=WEXITSTATUS(st);
25.             }
26.         }
27.         sprintf(buf, "Numero de procesos: %d\n", nprocs);
28.         write (1, buf, strlen(buf));
29.         exit(nprocs);
30.     }
31.     exit(0);
32. }
```

figura 1: Código de arbol.c

Suponiendo que todas las llamadas a sistema se ejecutan sin devolver ningún error, contesta a las siguientes preguntas de manera razonada.

- a) **(1 punto)** Dibuja la jerarquía de procesos que se genera al ejecutar el comando. En el dibujo asigna un identificador a cada proceso para usar en las preguntas posteriores.



Nombre alumno:

DNI:

- b) **(0,5 puntos)** ¿Qué proceso(s) ejecutará(n) la línea 17? Indica el valor que tiene la variable buf en la línea 17 para cada uno de ellos.

- c) **(0,5 puntos)** ¿Qué proceso(s) entra(n) en el bucle de la línea 21? Para cada uno de ellos, indica el número de iteraciones que hará.

- d) **(0,5 puntos)** ¿Qué procesos ejecutan la línea 31?

- e) **(0,5 puntos)** ¿Cuál el número máximo de procesos concurrentes que podemos tener?

- f) **(1 punto)** Para cada proceso, indica el valor de la variable nprocs justo antes de acabar la ejecución.

Nombre alumno:

DNI:

Limita (3,5 puntos)

Queremos hacer un programa que reciba dos parámetros: un nombre de comando y un número (mayor que cero) que será un límite de tiempo (le llamaremos timeout). El programa crea un proceso que mutará al comando. El proceso padre tiene que escribir un “.” por la salida cada segundo como muestra del tiempo pasado hasta llegar al timeout. El hijo queremos que muera directamente por la recepción del signal SIGALRM.

Nos dan el siguiente código que probamos y vemos que no funciona. Analízalo y contesta a las preguntas. Hemos eliminado el control de errores para reducir espacio. Asume que los programas a los que mutamos no realizan ninguna llamada a sistema de gestión de signals.

```
1. int t = 0;
2. void f_alarm(int s)
3. {
4.   char c='.';
5.   write(1,&c,sizeof(char));
6.   t += 1;
7.   alarm(1);
8. }
9.
10. int main(int argc, char *argv[])
11. {
12.   sigset_t m;
13.   int ret;
14.   struct sigaction sa;
15.   int timeout = atoi(argv[2]);
16.
17.   sigfillset(&m);
18.   sigprocmask(SIG_BLOCK, &m, NULL);
19.
20.   ret = fork();
21.   if (ret == 0){
22.     alarm(timeout);
23.     execlp(argv[1], argv[1], NULL);
24.   }
25.
26.   sigdelset(&m, SIGALRM);
27.   alarm(1);
28.   while(t < timeout) sigsuspend(&m);
29.
30. }
31.
```

- a) **(0,5 puntos)** Indica las 6 primeras llamadas a sistema que haría el padre y las que puedas del hijo (ya que no tienes el código)

	Padre	Hijo
1		
2		
3		
4		
5		
6		

Nombre alumno:

DNI:

- b) **(0,75 puntos)** Al ejecutarlo, nos sale el mensaje “Alarm clock” en la consola. ¿Quién escribe ese mensaje? ¿A qué se debe? ¿Cómo lo solucionarías? (Indica exactamente el código y donde lo pondrías)

- c) **(0,5 puntos)** También observamos que el hijo no termina al cabo de los X segundos que hemos usado para configurar la alarma. ¿A qué es debido? ¿Cómo lo solucionarías sin modificar ni el proceso padre ni el ejecutable a mutar?

- d) **(0,25 puntos)** Suponiendo que arreglamos el código para que haga lo que se pide, ¿habría algún conflicto entre la programación del alarm de las líneas 22 y 27?

- e) **(1 punto)** El código propuesto no tiene en cuenta que el programa al cual muta el hijo podría durar menos de los X segundos de timeout. Propón una modificación en la que, manteniendo la funcionalidad básica, detectemos que el proceso hijo ha terminado antes del timeout y terminemos la ejecución del padre escribiendo un mensaje antes de terminar “Timeout no agotado”. Propón una solución que no elimine el sigsuspend. Indica claramente qué código añadirías o modificarías. Usa los números de línea actuales como referencia.

Nombre alumno:

DNI:

- f) **(0,5 puntos)** Si el comando al que mutamos ejecuta el siguiente código al principio. ¿Explica que hace este código y cómo afectará a la funcionalidad básica que es conseguir que el proceso hijo termine al cabo de X segundos? ¿Qué modificaciones propones en el padre para forzar que el hijo termine al cabo de X segundos? Indica que código añadirías y en que líneas. ¿Podemos seguir usando SIGALRM para acabar con el hijo?

```
1 struct sigaction sa;
2 sa.sa_handler = SIG_IGN;
3 sigfillset(&sa.sa_mask);
4 sa.sa_flags = SA_RESTART;
5 sigaction(SIGALRM, &sa, NULL);
```