Nombre alumno:
DNI:

Primer parcial de teoría de SO

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

Gestión de procesos (6 *puntos*)

La figura 1 muestra el código del programa jerarquía (se omite el control de errores para facilitar la legibilidad del código):

```
1. /* jerarquia */
2.
3. main(int argc, char * argv[]){
4. int i, nprocs,ret,st;
5. char buf[80];
6. nprocs=atoi (argv[1]);
7. for (i=0;i<nprocs;i++){
8.     ret = fork();
9.     if (ret > 0) {
10.          sprintf(buf,"%d",nprocs-1);
11.          execlp("./proc","proc",buf,(char *)0);
12.      } else {
13.          sprintf(buf,"Iteracion %d\n",i);
14.          write(1, buf, strlen(buf));
15.          exit(i);
16.      }
17.  }
18.
19.  while (waitpid(-1,&st, 0) >0){
20.      if (WIFEXITED(st)) {
21.          sprintf(buf,"Acaba hijo %d\n",WEXITSTATUS(st));
22.          write(1,buf,strlen(buf));
23.      }
24.  }
25.  }
26.  }
```

figura 1: Código de jerarquia.c

Ponemos el programa en ejecución con el siguiente comando: ./jerarquia 3

Suponiendo que write, exec y exit se ejecutan sin devolver ningún error, contesta a las siguientes de manera razonada.

a) ***(1 punto)*** Dibuja la jerarquía de procesos que se genera al ejecuta el programa. En el dibujo asigna un número a cada proceso para las preguntas posteriores.

Nombre alumno:
DNI:

b) ***(0,75 puntos)*** ¿Qué valor máximo puede tomar la variable "i"? ¿Qué proceso(s) le dan ese valor?

c) ***(0,75 puntos)*** ¿Cuál es el valor máximo del grado de concurrencia que puede alcanzar este comando? ¿El código implementa un esquema secuencial o concurrente?

d) ***(0,75 puntos)*** ¿Qué procesos escriben el mensaje que empieza por "Acaba hijo" (write de la línea 22)? ¿Cuántas veces lo escribe cada proceso?

e) ***(0,75 puntos)*** Si un proceso padre llega a la llamada waitpid de la línea 19 cuando al menos un hijo ha acabado ya la ejecución, ¿qué efecto puede tener esta llamada sobre el estado en el que se encuentra el hijo?

f) ***(1 punto)*** ¿Qué líneas de este código provocarán siempre un cambio de modo de ejecución?

g) ***(1 punto)*** Supón que ejecutamos varias veces este código, ¿podemos garantizar que **todos** los cambios de modo de ejecución serán siempre en las mismas líneas?

Nombre alumno:
DNI:

Preguntas cortas (4 *puntos*)

Tenemos un SO con un ciclo de vida de procesos con los estados que hemos presentado en clase, incluyendo los estados "Sleeping" y "Stopped" en representación de Blocked. Es decir, como un SO Linux. Contesta razonando brevemente tus respuestas.

a) ***(0,75 puntos)*** Un proceso está en el estado de "Sleeping" y recibe un SIGALRM previamente reprogramado (una función que simplemente muestra un mensaje por pantalla). Explica las transiciones de estados que hará el proceso, indicando en qué momento ejecuta la función de tratamiento del signal.

b) ***(0,5 puntos)*** Si se ejecuta "alarm(1)" antes de "fork()", quién recibe el signal SIGALRM?

c) ***(0,5 puntos)*** En qué circunstancias se cambian cuáles son los signals que están bloqueados?

d) ***(0,75 puntos)*** Si reprogramamos SIGINT con el flag SA_RESETHAND, qué efecto tendrá en el siguiente código si pulsamos varias veces "Ctrl+C" mientras el proceso está bloqueado en la llamada bloqueante read?

```
while(1){
    ret = read(0, &c, 1);
    printf("El resultado es %d\n", ret);
}
```

e) ***(1 punto)*** Si ejecutamos la línea de comandos "#>./prog", qué veremos en la terminal? Si lo ejecutamos en múltiples ocasiones, siempre tendrá el mismo comportamiento?

Nombre alumno:

DNI:

```
                    /* prog */
1.  char buf[256];
2.  void func(int s){
3.      sprintf("Signal recibido\n");
4.      write(1, buf, strlen(buf));
5.  }
6.
7.  main(int argc, char * argv[]){
8.      struct sigaction sa;
9.      sigseg_t mask;
10.
11.     sigemptyset(&sa.sa_mask);
    sigfillset(&mask);
12.     sa.sa_flags=0;
13.     sa.sa_handler=func;
14.     sigaction(SIGALRM, &sa, NULL);
15.
16.     if ((pid=fork()) == 0){
17.         execlp("./miecho", "./miecho", NULL);
18.     }
19.     kill(pid, SIGALRM);
20. }
```

```
                /* miecho */
1.  char buf[256];
2.  void func(int s){
3.      sprintf("Signal recibido\n");
4.      write(1, buf, strlen(buf));
5.  }
6.
7.  main(int argc, char * argv[]){
8.      struct sigaction sa;
9.      sigseg_t mask;
10.
11.     sigfillset(&mask);
12.     sigprocmask(SIG_BLOCK, &mask,
    NULL);
13.     sigemptyset(&sa.sa_mask);
14.     sa.sa_flags=0;
15.     sa.sa_handler=func;
16.     sigaction(SIGALRM, &sa, NULL);
17.
18.     sprintf("%s ejecutado\n", argv[0]);
19.     write(1, buf, strlen(buf));
20. }
```

f) ***(0,5 puntos)*** Si tenemos todos los signals reprogramados para que ejecuten una función que no haga nada, enviando un signal desde el terminal al proceso, podemos llegar a mostrar el mensaje por pantalla?

```
sigfillset(&mask);
sigprocmask(SIG_BLOCK, &mask, NULL);
sigemptyset(&mask);
sigsuspend(&mask);
printf("Mensaje mostrado\n");
```