Nombre alumno:
DNI:

# Examen parcial de teoría de SO

**Justifica todas tus respuestas.** Las respuestas sin justificar se considerarán erróneas.

## Gestión de memoria (2 puntos)

Analiza el siguiente código. Responde a las siguientes preguntas de forma razonada. El código se ejecuta sobre un sistema operativo Linux, con tamaño de página de 4096 bytes, y ofrece la optimización Copy-On-Write. La región de código del programa ocupa 2400 bytes. Suponemos que el tamaño de las regiones para los datos viene dado únicamente por lo definido en este código.

```
#define MAXSIZE  4096

char arrayA[MAXSIZE];

int main (){
  char arrayB[MAXSIZE];
  char *arrayC;
  int ret;

  for (int i = 0; i < MAXSIZE; i++)
      arrayA[i] = 'a';

  arrayC = sbrk (MAXSIZE);
  ret=fork ();

  if (ret == 0)
    {
      for (int i = 0; i < MAXSIZE; i++)
         arrayB[i] = 'b';
    }
  else
    for (int i = 0; i < MAXSIZE; i++)
      arrayC[i] = 'c';
--------------------------PUNTO A
  sbrk (-1*MAXSIZE);
}
```

a) ***(0,5 puntos)*** En la siguiente tabla, indica en qué región se encuentran declaradas las variables y dónde se almacena su contenido.

| | Región de la variable | Región del contenido |
|---|---|---|
| **ArrayA** | | |
| **ArrayB** | | |
| **ArrayC** | | |

b) ***(0,5 puntos)*** Indica y justifica cuánta memoria física ocupa (en marcos de página) en el **punto A**

| | Num. frames | Justificación |
|---|---|---|
| **Código** | | |
| **Datos** | | |
| **Pila** | | |
| **Heap** | | |

c) ***(0,5 puntos)*** Identifica las invocaciones a llamada a sistema de gestión de memoria del código y sustitúyelas por llamadas de librería de lenguaje de gestión de memoria equivalentes:

| Llamada a sistema en el código | Función de librería |
|---|---|
| | |
| | |

Nombre alumno:
DNI:

d) ***(0,5 puntos)*** ¿Tendría sentido que el sistema operativo usara la optimización de carga bajo demanda a la hora de cargar y ejecutar este programa.

## Preguntas cortas (*2 puntos*)

a) ***(0,5 puntos)*** Podría usarse la librería de sistema de un SO Linux sobre arquitectura ARM en un sistema Linux de arquitectura Intel x64? Razona tu respuesta.

b) ***(0,5 puntos)*** Analiza el código. ¿Cuándo y bajo qué condiciones escribiría FIN por pantalla?

```
(…)
    sigfillset(&mask);
    sigdelset(&mask,SIGCHLD);
    sigsuspend(&mask);
    write(1,"FIN",3);
(…)
```

c) ***(0,5 puntos)*** Indica dos casos en que **la invocación** a la siguiente llamada a sistema podría provocar una excepción de gestión de memoria: **`write(1,"FIN",3);`**

**CASO 1:**

**CASO2:**

d) ***(0,5 puntos)*** Indica los casos en que un proceso abandona la CPU en una política de planificación apropiativa.

Nombre alumno:
DNI:

## Signals (3 *puntos*)

La figura 1 muestra el código del programa signal.c que causa la ejecución de un proceso padre y su hijo (se omite el control de errores para facilitar la legibilidad del código):

```
1.  /* signal.c */
2.  int alarma = 0;
3.  int pidh;
4.  void trat_signal(int signum) {
5.      if (signum==SIGALRM){
6.          alarma=1;
7.          kill(pidh,SIGKILL);
8.      }else
9.          alarm(0);
10. }
11. void f(int par) {
12.     // algun calculo que no afecta al ejercicio
13. }
14. main(int argc, char *argv[]) {
15.     struct sigaction s;
16.     char buf[80];
17.     sigset_t mask;
18.
19.     sigfillset(&mask);
20.     sigprocmask(SIG_BLOCK,&mask,NULL);
21.
22.     s.sa_flags=0;
23.     sigemptyset(&s.sa_mask);
24.     sigaddset(&s.sa_mask,SIGALRM);
25.     sigaddset(&s.sa_mask,SIGUSR1);
26.
27.     s.sa_handler=trat_signal;
28.     sigaction(SIGUSR1,&s, NULL);
29.     sigaction(SIGALRM,&s, NULL);
30.
31.     sigfillset(&mask);
32.     sigdelset(&mask,SIGUSR1);
33.     sigdelset(&mask,SIGALRM);
34.
35.     pidh = fork ();
36.
37.     if (pidh > 0) {
38.         alarm(1);
39.         sigsuspend(&mask);
40.         if (alarma==0) {
41.             f(getpid());
42.             sprintf(buf,"Proceso 1 acaba f\n");
43.             write(1,buf,strlen(buf));
44.         }
45.     } else {
46.         f(getpid());
47.         sprintf(buf,"Proceso 2 acaba f\n");
48.         write(1,buf,strlen(buf));
49.         kill(getppid(), SIGUSR1);
50.     }
51.     waitpid(-1,NULL,0);
52. }
```

*figura 1: Codigo de signal.c*

Suponiendo que las llamadas se ejecutan sin devolver ningún error inesperado, y que los únicos signals involucrados son los que se envían desde signal.c, contesta a las siguientes preguntas.

a) ***(0,5 puntos***) ¿Qué signals tendrá bloqueados cada proceso en la línea 36, justo después del fork? ¿Y en la línea 39, mientras se está ejecutando el sigsuspend?

Línea 36:

Línea 39:

Nombre alumno:
DNI:

b) ***(0,5 puntos)*** ¿Cuál(es) de las llamadas a sistema de este código puede(n) provocar que el proceso pase a estado bloqueado? Indica la llamada a sistema, qué procesos la ejecutan y de ellos cuáles pueden pasar a bloqueado.

c) ***(0,5 puntos)*** Suponiendo que la función f tarda mucho menos de 1 segundo en completarse, ¿qué signals recibirá cada proceso?

d) ***(0,5 puntos)*** Suponiendo que la función f tarda mucho más de 1 segundo en completarse, ¿qué signals recibirá cada proceso?

e) ***(0,5 puntos)*** Si quitamos las líneas 19 y 20, ¿podría afectar de alguna manera a la ejecución del proceso hijo? ¿Y a la del padre? Si es que no, justifica tu respuesta. Si es que sí, explica cómo.

f) ***(0,5 puntos)*** Si quitamos las líneas 31, 32 y 33, ¿podría afectar de alguna manera a la ejecución del hijo? ¿Y a la del padre? Si es que no, justifica tu respuesta. Si es que sí, explica cómo.

Nombre alumno:
DNI:

## Gestión de procesos (3 *puntos*)

La figura 2 muestra el código de los programas jerarquia1.c y jerarquía2.c (se omite el control de errores para facilitar la legibilidad del código):

```
1.  /* jerarquia1 */
2.
3.  main (int argc, char * argv[]) {
4.  int i,ret;
5.  char buf[80];
6.  for (i=0; i<argc-1; i++) {
7.     ret = fork();
8.     if (ret == 0) {
9.       execlp("./jerarquia2","jerarquia2",
    argv[i+1],(char *)0);
10.    }
11. }
12. while ((ret = waitpid(-1,NULL,0)) > 0 ) {
13.     sprintf(buf, "j1: Proc %d acaba\n", ret);
14.     write(1, buf, strlen(buf));
15. }
16. sprintf(buf, "j1: Final de %d\n",getpid());
17. write(1,buf,strlen(buf));
18. }
```

```
1.  /* jerarquia2 */
2.
3.  main(int argc, char *argv[]){
4.  int ret=0, i=0;
5.  char buf[80];
6.  while ((i<atoi(argv[1])) && (ret == 0)){
7.      ret = fork();
8.      i++;
9.  }
10. if (ret > 0){
11.     while ((ret = waitpid(-1,NULL,0))>0){
12.         sprintf(buf, "j2: Proc %d acaba\n", ret);
13.         write(1, buf, strlen(buf));
14.     }
15. }
16. sprintf(buf, " j2: Final de %d\n",getpid());
17. write(1,buf,strlen(buf));
18. }
```

*figura 2: Código de jerarquia1.c y jerarquia2.c*

Ambos programas se encuentran en el directorio actual de trabajo y ejecutamos el siguiente comando: ./jerarquia1 0 1 2

Suponiendo que fork, write y execlp 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 las preguntas posteriores.

b) ***(1 punto)*** Para cada mensaje que estos dos códigos pueden mostrar por pantalla, indica qué proceso(s) lo mostrarán y cuántas veces lo hará cada uno (la notación jerarquia1:L14 significa mensaje de la línea 14 del fichero jerarquia1.c). En la columna “Proc” pon el identificador que le hayas asignado a cada proceso en tu dibujo de la jerarquía. Usa sólo las filas que necesites.

Nombre alumno:
DNI:

| **jerarquia1**:L14 | | **jerarquia1**:L17 | |
|---|---|---|---|
| Proc | Cuántas veces | Proc | Cuántas veces |
| | | | |
| | | | |
| | | | |
| | | | |

| **jerarquia2**:L13 | | **jerarquia2**:L17 | |
|---|---|---|---|
| Proc | Cuántas veces | Proc | Cuántas veces |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Justificación**

c) ***(0,5 puntos)*** ¿Podemos saber cuál será el último mensaje que aparecerá en pantalla? Si es así indica el mensaje y el proceso que lo mostrará. En cualquier caso, justifica tu respuesta.

d) ***(0,5 puntos)*** ¿Cuál es el grado máximo de concurrencia que podemos tener?