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	Datos	Datos
ArrayB	Pila	Pila
ArrayC	Pila	heap

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

	Num. frames	Justificación
Código	1 frame	El código se comparte entre padre e hijo. Los 2400 bytes caben en 1 frame.
Datos	1 frame	El arrayA no se actualiza entre los puntos A y B así que también es compartido. 4096 bytes es un frame
Pila	4 frames	En la pila tenemos arrayB (1 frame) y las variables arrayC y ret (caben en otro frame). Total 2 frames. El proceso hijo actualiza el arrayB (el padre no lo hace) por tanto se debe duplicar el frame que lo contiene. La variable **ret** tiene valores diferentes para padre e hijo, y por tanto la página se duplica.
Heap	2 frames	Se piden 4096bytes que son 1 frame que después del fork estará compartido. El padre actualiza el array C, el hijo no. Se duplica el frame compartido 2 frames

Nombre alumno:
DNI:

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
arrayC=sbrk(MAXSIZE);	arrayC=malloc(MAXSIZE);
sbrk(-1*MAXSIZE);	free(arrayC);

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.

No tendría sentido. El código cabe completamente en una única página, que es la unidad mínima de asignación de memoria. Sería más optimo cargar todo el código en la página que no ir haciendo cargas parciales de código a medida que se necesite.

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.

No podría. La librería de sistema incluye el código de las llamadas a sistema. El código de las llamadas a sistema es dependiente de instrucciones y registros específicos de cada arquitectura (en diferentes arquitecturas hay diferentes implementaciones del TRAP)

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);
(…)
```

Cuando se reciba un SIGCHLD, que debe estar previamente capturado y su rutina de atención no incluya la finalización del proceso.

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: En caso de que el programa esté compilado estáticamente y haya que cargar la rutina de write porque la optimización de carga bajo demanda no la haya cargado previamente.
CASO2: En caso de que el código de write esté en el área de SWAP

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

Nombre alumno:
DNI:

- Cuando el proceso acabe explícitamente o por la recepción de un signal cuyo comportamiento haga acabar el proceso o pararlo
- Cuando ejecute una llamada a sistema bloqueante.
- Cuando agote su ráfaga de CPU

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.

Nombre alumno:
DNI:

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: Los dos procesos tienen todos bloqueados: el padre los bloquea en la línea 17 y el hijo hereda la máscara de los signals bloqueados
Línea 39: Esta línea solo la ejecuta el padre y solo afecta a su máscara. Los tendrá bloqueados todos menos el SIGUSR1 y el SIGALRM, que es la máscara que se pasa como parámetro al sigsuspend

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.

El sigsuspend de la la línia 3 solo lo ejecuta el padre y puede hacer que se bloquee si no tiene pendiente de tratar SIGUSR1
El waitpid de la línia 51 lo pueden ejecutar los dos procesos pero solo puede bloquear al padre ya que el hijo no tiene hijos

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

Si tarda menos de 1 segundo, el proceso hijo enviarà el SIGUSR1 a su padre que desactivarà el temporitzador y ejecutará f. Y recibirá el SIGCHLD cuando su hijo muera.

El hijo no recibe ningún signal

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

El padre recibirá el SIGALRM que causa el alarm(1) de la línea 38 y el SIGCHLD. El hijo el SIGKILL que le envía el padre en la línea 7

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.

Nombre alumno:
DNI:

Al hijo no, ya que el único signal que puede recibir el hijo es el SIGKILL y ese no se bloquea.

Al padre solo podria afectarle si el hijo le mandara el signal antes de entrar en el sigsuspend. En ese caso el padre se quedaria bloqueado en el sigsuspend hasta que saltara la alarma y el comportamiento sería como si la función f hubiera tardado más de 1 segundo.

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.

Al hijo no, ya que su código no utiliza la variable mask

Al padre le afectaria porque dentro del sigsuspend tendría todos los signals bloqueados (la variable mask se inicializa 19 con todo a 1) y nunca saldría del sigsuspend.

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. }
```

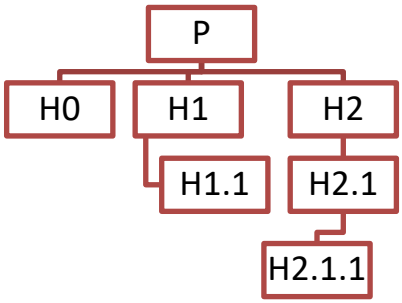
figura 2: Código de jerarquia1.c y jerarquia2.c

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.

Nombre alumno:
DNI:



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.

jerarquia1:L14		**jerarquia1**:L17	
Proc	Cuántas veces	Proc	Cuántas veces
P	3	P	1

jerarquia2:L13		**jerarquia2**:L17	
Proc	Cuántas veces	Proc	Cuántas veces
H1	1	H0	1
H2	1	H1	1
H2.1	1	H1.1	1
		H2	1
		H2.1	1
		H2.1.1	1

Justificación: Los mensajes de jerarquía1 solo los muestra el proceso inicial porque el resto muta en el exec de la línea 9. El de la línea 14 lo muestra una vez por cada hijo que muere. En este caso, se crean 3 procesos porque es el número de parámetros que pasamos a jerarquia1. y el de la línea 17 una sola vez.

P no muestra los mensajes de jerarquía 2 porque no ejecuta ese código. Lo harán sus 3 hijos y la jerarquía que éstos creen. El mensaje de la línea 14 lo muestran los procesos que tienen algún hijo (no lo muestran ni H0, ni H1.1 ni H2.1.1). El de la línea 17 lo muestran todos los procesos que ejecutan jerarquia2.

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.

Nombre alumno:
DNI:

Si, será el mensaje de la línea 17 de jerarquía1 que muestra P. Este mensaje se mostrará cuando todos sus hijos hayan acabado, y a su vez sus hijos esperan a que todos los suyos acaben.

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

El esquema de creación en los dos códigos es concurrente. Así que el grado máximo será la cantidad de procesos de la jerarquía: 7