Apartado	1	2	3	4	5	6	7	8
Puntuación								

□ No Presentado

EXAMEN DE SISTEMAS OPERATIVOS (Grado en Ing. Informática) 9/7/2012.

APELLIDOS Y NOMBRE: .....

Justificar todas las respuestas. Poner apellidos y nombre en todas las hojas y graparlas a este enunciado. Tiempo total = 2 horas.

1. (**0.75 puntos**) **(a)** Descríbase brevemente la estructura en capas del software de entrada/salida.

   - software de usuario
   - software independiente del dispositivo
   - manejador de dispositivo (device driver)
   - manejador de interrupciones

   (b) ¿En cuál de dichas capas se encontraría la función `printf`?
   software de usuario
   (c) Suponiendo que se quiere añadir soporte para unidades SCSI, ¿cuál (o cuáles) de las capas de software de E/S se verían afectadas?
   manejador de dispositivo (*device driver*)

2. (**1 punto**) En un sistema con paginación y sin memoria virtual, las direcciones lógicas de 32 bits se dividen en 16 para el número de página, y 16 para el offset. En un momento dado, tenemos en memoria 32 copias de un proceso de 160 KB.
   (a) ¿Qué tipo(s) de fragmentación tendríamos?
   Interna: cada proceso no ocupa un múltiplo exacto del tamaño de página. Por otro lado, en un sistema con paginación no existe fragmentación externa.
   (b) ¿Cuánta memoria se desperdicia?
   Con 16 bits de offset el tamaño de página será 2^{16} = 64 KB. Cada proceso ocupa 160 KB pero necesita 3 páginas completas 3×64 = 192 KB. Es decir, cada proceso desperdicia 192-160=32 KB. Como hay 32 procesos 32×32 KB = 1024 KB = **1 MB**.
   (c) ¿Cuánto se desperdiciría si, con esos mismos procesos, tuvieramos 20 bits para el número de página y 12 para el offset?
   Con 12 bits de offset el tamaño de página será 2^{12} = 4 KB. Como 160 es múltiplo de 4, cada proceso ocupa 40 páginas completas y **no se desperdicia memoria por fragmentación interna** en este caso.

3. (**1 punto**) En un sistema de paginación bajo demanda, un proceso genera la secuencia de referencias de página de la figura de abajo. Complete el desarrollo para el algoritmo de reemplazo *FIFO segunda oportunidad* (la comilla indica: bit de referencia = 1)

Secuencia	a	b	c	d	e	c	a	d	b	d	e	c
páginas residentes	a'	a'	a'	a'	e'	e'	e'	e'	e	e	e'	e
	-	b'	b'	b'	b◂	b◂	a'	a'	a	a	a	c'
	-	-	c'	c'	c	c'	c'◂	c'◂	b'	b'	b'	b'◂
	-	-	-	d'	d	d	d	d'	d◂	d'◂	d'◂	d
¿Fallo?	F	F	F	F	F		F		F			F

4. (**0.75 puntos**) Considérense los siguientes programas y supóngase que incluyen todas las sentencias `#include` necesarias para su compilación.

```
/*****programa P1********/
main() {
  unsigned long veces=2000000000;
  unsigned long bucle=2000000000;
  while (--veces)
    while (--bucle);
}
```

```
/*****programa P2********/
main() {
  unsigned long veces=2000000000;
  unsigned long bucle=2000000000;
  while (--veces)
    while (--bucle)
      getpid();
}
```

```
/*****programa P3 ********/
double funcion (double x) {
  double raiz, seno;
  raiz=sqrt(x);
  seno=sin(raiz);
  return seno;
}

main() {
  unsigned long veces=2000000000;
  unsigned long bucle=2000000000;
  float x;
  while (--veces)
   while (--bucle)
    x= (float)
      funcion((double)(bucle *veces));
}
```

Señalar la respuesta correcta:

- ☐ P1, P2 y P3 se ejecutan todo el tiempo en modo kernel
- ☐ P1, P2 y P3 se ejecutan todo el tiemo en modo usuario
- ☒ P1, P2 y P3 se ejecutan parte del tiempo en modo usuario y parte en modo kernel
  La llamada al sistema para terminar, que realizan todos los procesos, es en modo kernel, y cuando un proceso se crea, el proceso creado completa la llamada de creación en modo kernel. Además los cambios de contexto se realizan en modo kernel. En este caso concreto, P2 ejecuta tambien la llamada `getpid()` pero eso no implixa que P1 y P3 no ejecuten nada en modo kernel.
- ☐ P1 y P3 se ejecutan todo el tiempo en modo usuario y P2 todo el tiempo en modo kernel
- ☐ P1 y P3 se ejecutan todo el tiempo en modo usuario y P2 parte del tiempo en modo kernel y parte del tiempo en modo usuario
- ☐ Depende de las credenciales de los procesos creados para ejecutar dichos programas
- ☐ Ninguna de las anteriores: (concrétese)

5. (**1.5 puntos**) Un sistema de archivos tipo *system V* tiene un tamaño de bloque de 8 Kbytes e *inodos* con 10 direcciones directas de bloques, una indirecta simple, una indirecta doble y una indirecta triple. Además, utiliza direcciones de bloques de 8 bytes. Consideremos el fichero "direcciones" en el directorio `/home/usr1`, con un tamaño de 9 MBytes. **Cada respuesta solo se puntuará si es correcta.**

   (a) Calcular cuántos bloques de disco son necesarios para representar ese archivo de 9 MBytes. Nº bloques de datos: 1152 Nº de bloques de índices: 3

(b) Un proceso abre el archivo `fd=fopen("/home/usr1/direcciones", O_RDONLY);` suponiendo que tiene permiso de lectura, y que las caches de datos e *inodos* están inicialmente vacías y que la entrada `home` está en el quinto bloque del directorio raíz, calcular cuántos accesos a disco son necesarios, como mínimo, para esa apertura.

Nº mínimo de accesos		Nº mínimo de accesos al área de datos		Nº mínimo de accesos a la lista de inodos
[8]	=	[7]	+	[1]

(c) Tras los supuestos anteriores, una vez abierto el archivo, lo primero que realiza el proceso es `lseek(fd, 2097152, SEEK_SET)`. Calcula cuántos bloques habría que leer ahora de disco para la operación `c=fgetc(fd)` suponiendo el Buffer Cache vacío. (Nota: $2097152 = 2 * 2^{20}$). Nº bloques que es necesario leer = [2]

6. (**1 punto**) Un intérprete de comandos mantiene una lista de procesos en segundo plano que se encarga de lanzar. Dígase si es correcta la siguiente función para crear un proceso (se supone que *args* está debidamente terminado a `NULL`, y que la función `meterListaProcesosSegundoPlano` no contiene errores)

```
void CrearProceso (char *args[], int primerPlano) {
   pid_t pid;
   int estado;

   if (args[0]==NULL){
        printf ("Nada que ejecutar");
        return;
   }
   if ((pid=fork())==-1){
        perror ("fallo en fork()");
        return;
   }
   if (pid==0){
       execvp(args[0],args);
       perror ("fallo en execv");
       return;
   }
   else if (primerPlano)
             wait(NULL);
        else meterListaProcesosSegundoPlano(pid,args);
}
```

- ☐ **Es correcta**: en ese caso, modifíquese para que, cuando termine un proceso en primer plano, indique si ha sido normalmente (valor devuelto) o debido a una señal (indicando la señal que los ha terminado).
- ☒ **Tiene errores**: en ese caso indíquese cuáles son y corríjanse.
  Es incorrecta: Dos errores
  - Después de `perror("fallo en execv");` debe haber una llamada al sistema `exit()` y no un return, puesto que el proceso ya ha sido creado y return simplemente haría que hubiese dos copias del shell en ejecución.
  - Al haber procesos en segundo plano, `wait(NULL);` debe sustituirse por `waitpid(pid, NULL,0);` (o `waitid` o `wait4`) puesto que la llamada `wait` finaliza la espera cuando termina *cualquier* proceso hijo, mientras que en este caso queremos esperar por el que hemos lanzado en primer plano.

7. (**1 punto**) En un sistema con un único procesador, existen dos procesos que presentan la siguiente secuencia de ejecución:

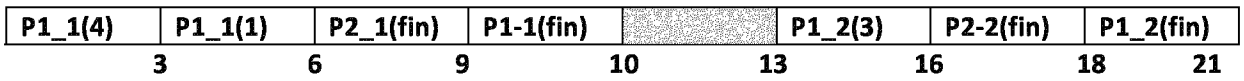
Proceso	Momento de llegada	Ráfaga CPU	Ráfaga E/S	Ráfaga CPU
P1	0	7	3	6
P2	5	3	5	2

Analizar los tiempos de retorno medio de para los siguientes algoritmos de planificación de la CPU: **(a)** RR de quantum 3; y **(b)** SRTF.

RR

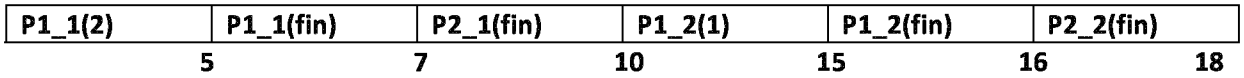
Formato de la solución: Px_y(z)

x→ proceso, y → ráfaga, z → tiempo pendiente al final de la ráfaga analizada



Trm=(21-0)+(18-5)/2=17

SRTF



Trm=(16-0)+(18-5)/2=14,5

8. (**1 punto**) (**EQUIVALE A TGR**): Un sistema con paginación tiene direcciones virtuales de 16 bits de las que los 4 bits más significativos corresponden al número de página y los 12 menos significativos al desplazamiento. Las entradas de la tabla de páginas están formadas por 8 bits. Los 4 menos significativos son el número de marco físico donde está la página, el bit 7 (el más significativo) es el bit de presencia y el 6 el de referencia. Los enteros son de 4 bytes. El código de un proceso y la tabla de páginas del proceso son las siguientes:

```
#define MAX 4096
int a[MAX];
main()
{
  int i;
  for(i=0;i<MAX;i++)
    a[i]=0;
}
```

```
0x00
0x4c
0x86
0x5b
0xa3
0xfa
0xcb   //página 0
```

Sabiendo que la matriz comienza en la dirección lógica `0x1000`: **(a)** ¿Cuántos marcos de memoria física tiene asignado el proceso en este momento?

4 marcos.

¿Por qué?

Los bits de presencia a 1 indican las páginas en memoria.

(b) ¿Qué porcentaje de la matriz está en la memoria y qué direcciones de memoria física ocupa?

```
0 0 00  0000
0 1 00  1100
1 0 00  0110 --+
0 1 01  1011   |------ marcos de página
1 0 10  0011   |       de la tabla de datos
1 1 11  1010 --+
1 1 00  1011
```

El 75 % a partir de las posiciones: 0xA000 + 4K, 0x3000 + 4K y 0x6000 + 4K.
Están en la memoria las páginas virtuales 0, 1, 2 y 4 (bit de presencia a 1)
Pertenecen a la tabla de datos las páginas 1, 2 y 4
La página 1 ocupa el marco 10 (1010) 0xA000 + 4K
La página 2 ocupa el marco 3 (0011) 0x3000 + 4K
La página 4 ocupa el marco 6 (0110) 0x6000 + 4K