Apartado	1	2	3	4	5	6	7	8
Puntuación								

EXAMEN DE SISTEMAS OPERATIVOS (Grado en Ing. Informática) 24/5/2012.

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)** Describe brevemente la estructura en capas del software de entrada/salida.

   - Software de nivel de usuario: interfaz con el usuario, formateo de datos.
   - Software independiente del dispositivo: asignación de espacio, control de privilegios, uso de caché, etc
   - Manejador de dispositivo (device driver): se comunica con el controlador de dispositivo. Única parte del S.O. que conoce los detalles físicos del dispositivo.
   - Manejador de interrupciones: gestiona las interrupciones que genere el dispositivo.

   (b) ¿En cuál de dichas capas se ubicaría el planificador de disco?

   En el manejador de dispositivo (device driver).

   (c) ¿En cuál de ellas se realiza la conversión de un *offset* de fichero (ej. para llamadas `read` o `write`) a número de *bloque* del sistema de ficheros y *offset* en dicho bloque?

   En el software independiente de dispositivo.

2. (**0.75 puntos**) Sean A y B dos direcciones virtuales de distintos procesos ¿Cuál (o cuáles) de las siguientes afirmaciones son **posibles** en un sistema con paginación?

   (a) A es distinta de B y sus respectivas direcciones físicas también son distintas entre sí.

   (b) A es distinta de B, pero ambas corresponden a la misma dirección física.

   (c) A y B son iguales, pero sus respectivas direcciones físicas son distintas entre sí.

   (d) A y B son iguales y sus respectivas direcciones físicas también coinciden.

   Son todas posibles. Tanto (a) como (c) son la situación habitual, ya que a las direcciones **virtuales** de un proceso y a las de otro normalmente (si no se comparte memoria) se les hace corresponder zonas de memoria física **distintas**. Las situaciones (b) y (d) se podrían dar en el caso de que ambos procesos estén usando una zona de memoria compartida: en el caso (b), "colocada" (attached) en distintas direcciones virtuales, y en el caso (c), casualmente colocada en las mismas posiciones virtuales.

3. (**1 punto**) En un sistema de paginación bajo demanda, un proceso genera la secuencia de referencias de página de la figura de abajo, donde también se muestran las páginas residentes en memoria.

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

(a) Completar la figura: contenido de los marcos y fallos de página que falten.

(b) ¿Qué algoritmo de reemplazo (visto en clase) está siendo usado?

Será LRU, dado que si fuese FIFO, en el instante 7, el fallo de página provocado por *a* tendría que haber reemplazado a *b*, que fue el que entró antes, de los que estaban en memoria en ese momento. Tampoco es el algoritmo óptimo, puesto que en el instante 5, la entrada de *e* debería haber provocado el reemplazo de *d*, que es la referencia que más tarde aparece en el futuro.

Con *segunda oportunidad* el desarrollo completo sería:

a	b	c	d	e	b	a	c	e	b	d	a
a'	a'	a'	>a'	e'	e'	e'	>e'	>e'	>e'	d'	d'
>-	b'	b'	b'	>b	>b'	b	b	b	b'	>b	>b
-	>-	c'	c'	c	c	a'	a'	a'	a'	a	a'
-	-	>-	d'	d	d	>d	c'	c'	c'	c	c
F	F	F	F	F		F	F			F	

4. (**1 punto**) En un sistema UNIX un proceso del `root` se ejecuta en modo kernel ...

(a) Siempre

(b) Nunca

(c) Si lo especifica el usuario `root`

⇒ **(d)** Ninguna de las anteriores

Desde el punto de vista de modo usuario versus modo kernel, un proceso del `root` es un proceso de usuario como otro cualquiera. Entrará en modo kernel cuando sea necesario para la operación que esté realizando (llamada al sistema, excepción, inicio o final del proceso, etc).

¿Que significa que el kernel de UNIX sea reentrante?

Varios procesos pueden estar ejecutando concurrentemente funciones del kernel.

¿Qué condiciones son necesarias para que el kernel pueda ser reentrante?

(1) Código del kernel de sólo lectura; (2) Protección de accesos concurrentes a las estructuras de datos del kernel; (3) Cada proceso tendrá su propia pila de ejecución para rutinas del kernel.

5. (**1 punto**) Sea el siguiente código

```
void CreaXterms (int n) {
  int i;    pid_t pid;     char *args[]={"xterm", NULL};
```

```
  for(i=0; i<n;i++)
    if ((pid=fork())==0)
     execvp ("xterm",args);
}
```

¿Cuántos procesos se crearán con `fork` invocando `CreaXterms(4)` en cada uno de los siguientes casos? **(a)** `xterm` está en el `PATH`; y **(b)** `xterm` no está en el `PATH`.

a *execvp()* no crea procesos, sólo hace que un proceso ya creado con *fork()* ejecute otro programa (reemplaza su código). Por tanto se crearían 4 procesos (uno en cada iteración del bucle), cada uno de los cuales ejecutaría un `xterm`.

b Se crean 15 procesos. EXPLICACION: al no estar `xterm` en el PATH, *execvp()* fallará y por tanto no reemplaza el código del proceso creado mediante *fork()* con el del `xterm`, consecuentemente todos los procesos creados siguen ejecutando el bucle, creando a su vez más procesos.

Sea P el proceso que llama a `CreaXterms(4)`. En la primera iteración ($i = 0$) crea un proceso, que denominaremos P_0, que intenta ejecutar `xterm` mediante *execvp()*. Como *execvp()* falla (`xterm` no está en el PATH) tenemos dos procesos que van a ejecutar la segunda iteración del bucle.

La segunda iteración del bucle ($i = 1$) es ejecutada por P y P_0, y cada uno de ellos creará un proceso mediante *fork()*: P_1 y P_{01}.

La tercera iteración del bucle ($i = 2$) es ejecutada por 4 procesos P, P_0, P_1 y P_{01} y cada uno de ellos creará un proceso mediante *fork()*: P_2, P_{02}, P_{12} y P_{012}

La cuarta iteracción del bucle ($i = 3$) es ejecutada por 8 procesos: P, P_0, P_1, P_{01}, P_2, P_{02}, P_{12} y P_{012} y cada uno de ellos creará un proceso mediante *fork()*: P_3, P_{03}, P_{13}, P_{013}, P_{23}, P_{023}, P_{123} y P_{0123}

En total tenemos 16 procesos: P y los 15 procesos que se han ido creando en las distintas iteacciones del bucle: P_0, P_1, P_{01}, P_2, P_{02}, P_{12}, P_{012}, P_3, P_{03}, P_{13}, P_{013}, P_{23}, P_{023}, P_{123} y P_{0123}, de los cuales

- cuatro son hijos de P (P_0, P_1, P_2, P_3), creados en cada iteración del bucle que ejecuta P;
- tres son hijos de P_0 (P_{01}, P_{02}, P_{03}), creados en cada iteracción del bucle que ejecuta P_0,
- hay dos hijos de P_1 (P_{12}, P_{13}) y dos hijos de P_{01} (P_{012}, P_{013}), que corresponden al las iteracciones del buche que ejecutan dichos procesos
- hay un un hijo de cada uno de los procesos que solo ejecutan la última iteracción del bucle. Dichos procesos son P_2, P_{02}, P_{12} y P_{012} y los hijos creados en esta última iteracción son P_{23}, P_{023}, P_{123} y P_{0123}, respectivamente

Si quisiesemos que en el caso de que `xterm` no estuviese en el PATH no se crease ningún proceso, el código podría ser

```
void CreaXterms (int n) {
  int i;
  pid_t pid;
  char *args[]={"xterm", NULL};
```

```
  for(i=0; i<n;i++)
    if ((pid=fork())==0)
     if (execvp ("xterm",args)==-1){
           perror("imposible ejecutar xterm");
           exit (255);
        }
}
```

6. (**1.5 puntos**) Un sistema de archivos tipo *system V* tiene un tamaño de bloque de 2 Kbytes e *inodos* con 12 direcciones directas de bloques, una indirecta simple, una indirecta doble y una indirecta triple. Además, utiliza direcciones de bloques de 4 bytes. Consideremos el fichero "direcciones" en el directorio `/home/usr1`, con un tamaño de 5 MBytes. **Cada respuesta solo se puntuará si es correcta.**

   (a) Calcular cuántos bloques de disco son necesarios para representar ese archivo de 5 MBytes. Nº bloques de datos: 2560 Nº de bloques de índices: 6

   (b) Calcular qué número de bloque lógico corresponde al inodo del raíz (se comienza a contar en el bloque lógico 0), y cuál bloque lógico al inodo del fichero direcciones, suponiendo lo siguiente:

   - El número de inodo del "/" es el 2, y al fichero direcciones le corresponde el inodo núm. 325 (los inodos se comienzan a numerar a partir de 1).
   - El tamaño de un inodo es de 64 bytes.
   - El boot ocupa un bloque y el *superbloque* 7 bloques.

   Nº bloque lógico inodo "/" = 8 Nº bloque lógico inodo direcciones = 18

   (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 = 3

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

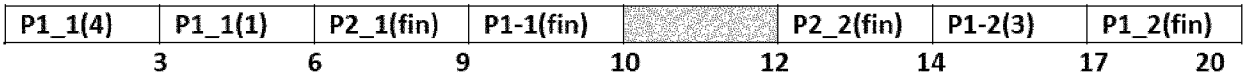
Proceso	**Momento de llegada**	**Ráfaga CPU**	**Ráfaga E/S**	**Ráfaga CPU**
P1	0	7	3	6
P2	5	3	3	2

Presenta el diagrama de evolución de estos procesos en la CPU para los algoritmos de planificación Round Robin (quantum=3) y SRTF y calcula los tiempos de retorno medio para ambos.

RR

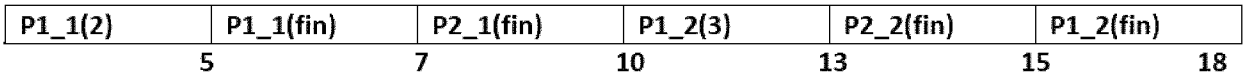
Formato de la solución: Px_y(z)

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



Trm=(20-0)+(14-5)/2=14,5

SRTF



Trm=(18-0)+(15-5)/2=14

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 `0x2000`: **(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   |------ marcos de página
0 1 01  1011   |       de la tabla de datos
1 0 10  0011 --+
1 1 11  1010
1 1 00  1011
```

El 50 % a partir de las posiciones: 0x3000 + 4K y 0x6000 + 4K.
Está en la memoria la página virtual 2 (bit de presencia a 1)
La página 2 ocupa el marco 3 (0011) 0x3000 + 4K
La página 4 ocupa el marco 6 (0110) 0x6000 + 4k