Apartado	1	2	3	4	5	6	7	☐ TGR
Puntuación								

☐ No Presentado

EXAMEN DE SISTEMAS OPERATIVOS (Grado en Ing. Informática) 11/7/2013.

APELLIDOS Y NOMBRE: .....

Justificar todas las respuestas. Poner apellidos y nombre en todas las hojas y graparlas a este enunciado. Tiempo total = **2 horas 30 minutos**.

1. (**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 5,5 MBytes.

   Responder a lo siguiente (**Cada respuesta sólo se puntuará si es correcta**):

   (a) Calcular cuántos bloques de disco son necesarios para representar ese archivo de 5.5 MBytes. Discriminar cuántos bloques son de datos y cuántos de índices.

   Número de bloques de datos: [2816]

   Número de bloques de índices: [7] (0.4 puntos)

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

   Número mínimo de accesos: [6] =

   Número mínimo de accesos al área de datos: [5] +

   Número mínimo de accesos a la lista de inodos: [1] (0.4 puntos)

   (c) Calcular qué número de bloque lógico en el sistema de ficheros 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. 66 (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.

   Número bloque lógico inodo "/" = [8]

   Número bloque lógico inodo fichero `direcciones` = [10] (0.4 puntos)

   (c) Tras los supuestos anteriores, una vez abierto el archivo, lo primero que realiza el proceso es `lseek(fd, 4194304, SEEK_SET)`. Calcula cuántos bloques habría que leer ahora de disco para la operación `c=fgetc(fd)`, suponiendo el BC vacío. (Nota: $4194304 = 4 \cdot 2^{20}$ y `SEEK_SET` indica que el desplazamiento se considera a partir del origen del fichero).

   Número bloques a leer = [3] (0.3 puntos)

2. (**1.5 puntos**) Suponga un proceso con 8 segmentos de igual tamaño y que el sistema construye una tabla de páginas para cada segmento con 8 entradas en la tabla para cada segmento, es decir, el sistema combina segmentación y paginación. El tamaño de la página es de 2Kb.

   Contéstese razonando brevemente a las siguientes cuestiones

   a) ¿Cuál es el tamaño máximo de cada segmento? 16 Kb

      Cada segmento puede tener hasta 8 páginas de 2Kb cada una.

   b) ¿Cuál es el tamaño máximo del espacio de direcciones lógico del proceso? 128 Kb

      Un proceso puede tener hasta 8 segmentos de 16Kb cada uno.

   c) Suponga un acceso al elemento en la posición de memoria física 00020AB0 ¿Cuál es el formato de la dirección lógica que generó este acceso?

      Dirección lógica = 3 bits para indicar el segmento, 3 bits para indicar la entrada en la tabla de páginas del segmento, 11 bits para el offset (las páginas son de 2Kb $=2^{11}$).

   d) ¿Cuál es el número de frame (marco) y offset de la dirección de memoria física?
      65 número de frame
      2B0 offset

      Dirección física = 0000 0000 0000 0010 0000 1010 1011 0000. Por tanto los últimos 11 bits corresponden al offset, es decir 010 1011 0000 = 2B0. Los 21 bits más significativos son el nmero de frame = 65

   e) ¿Cuál es el tamaño máximo del espacio de direcciones físicas del proceso? 4 Gb

      2^{32} = 4 Gb, o lo que es lo mismo 2^{21} frames de 2Kb cada una.

3. (**0.25 puntos**) ¿Es posible un sistema de paginación sin memoria virtual? Si/No, ¿Por qué?

   Sí, es posible. Todas las páginas del proceso tienen que estar en memoria. La paginación es una forma de transformar las direcciones lógicas en direcciones físicas y evitar el problema de la fragmentación externa de la memoria, con un pequeño coste de fragmentación interna.

4. (**0.25 puntos**) En un sistema de memoria virtual con paginación el conjunto residente de un proceso y el conjunto de trabajo (working set) de un proceso son lo mismo. Conteste a la afirmación anterior Si/No y razone la respuesta.

   No son lo mismo. El conjunto residente es el conjunto de páginas del proceso que están en memoria y el conjunto de trabajo es el conjunto de páginas que han sido referenciadas recientemente (en un cierto intervalo temporal).

5. (**1 punto**) Modifíquese el código del programa de abajo para que todo lo que se escribiría en el fichero "**salida.txt**" durante la llamada a la función **realiza_trabajo_1** se muestre por la salida estándar (STDOUT_FILENO), y que la función **realiza_trabajo_2** siga almacenando sus datos en el fichero "**salida.txt**".

```
#include "includes.h"
int main(){
    int fd;
    fd= open("salida.txt",O_WRONLY|O_CREAT, 0664);
```

```
    //código de redirección
    int copia = dup(fd);
    close(fd);
    dup(STDOUT_FILENO);
```

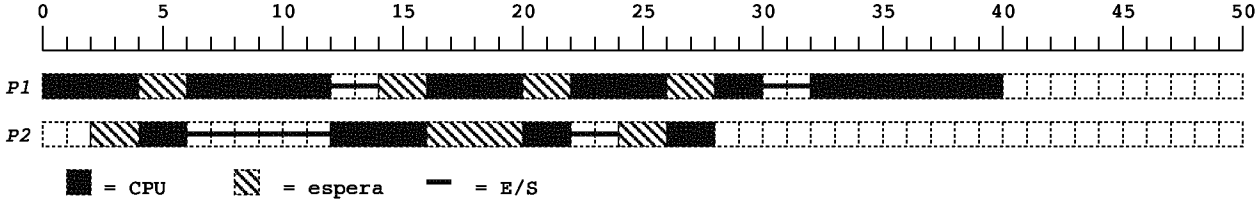
```
    realiza_trabajo_1(fd);
```

```
    //código para restaurar redirección
    close(fd);
    dup(copia);
    close(copia);
```

```
    realiza_trabajo_2(fd);
    close(fd);
    }
```

   Nota: No es posible modificar las funciones **realiza_trabajo_1** y **realiza_trabajo_2**, ya que no disponemos de su código fuente.

6. (**1 punto**) Tenemos dos procesos, $P1$ y $P2$, que inician su ejecución en los instantes $T_1 = 0$, y $T_2 = 2$ respectivamente (la unidad temporal es el milisegundo). Las secuencias de ráfagas de cada proceso son $P1 = (\underline{10}, 2, \underline{10}, 2, \underline{8})$, $P2 = (\underline{2}, 6, \underline{6}, 2, \underline{2})$ donde los números subrayados representan tiempo de CPU y el resto es E/S. Dibuja el diagrama de ejecución del algoritmo Round Robin con quantum $q = 4$. Calcula el **tiempo de espera** total para cada proceso.



Tiempo de espera $P1$= [8] ms Tiempo de espera $P2$= [8] ms

7. (**1,5 puntos**) Se desea implementar en un intérprete de comandos la orden *duplo* que crea dos procesos que ejecutan en segundo plano el mismo programa (con los argumentos) que se le pasan como parámetro. Por ejemplo

```
#) duplo xterm -e bash
```

crearía dos procesos y cada uno de ellos ejecutaría, en segundo plano, *xterm -e bash*. Suponemos que *args* es un array de punteros debidamente terminado a `NULL`, obtenido de trocear la cadena de entrada, y que en el shell, hay algo equivalente a lo siguiente:

```
.....
if (!strcmp(args[0],"duplo"))
  cmd_duplo(args+1);
....
```

Se propone la siguiente implementación de dicho comando, *cmd_ duplo*, del shell. Se supone además que la función *InsertarListaProcesos* es correcta.

```
int Ejecutar (char * args[]) {
  if (args[0]==NULL){
      errno=EFAULT;
      return -1;
  }
  return (execv (args[0],args));
}

void EjecutarProceso (char * args[],int primerplano) {
  pid_t pid;
  if ((pid=fork())==-1) {
      perror ("Imposible crear proceso"); return;
  }
  if ((pid==0){
      Ejecutar(args);
      perror ("Fallo en Ejecutar");
      exit(255);
  } else if (primerplano)
           waitpid(pid, NULL, 0);
         else
           InsertarListaProcesos (pid,args);
}

void cmd_duplo( char * args[]) {
  EjecutarProceso(args,0);
  EjecutarProceso(args,0);
}
```

Elíjase cual de las siguientes respuestas es correcta y respóndase adecuadamente:

(a) **La implementación es correcta**: modifíquese para que la ejecución de ambos procesos sea en primer plano: los dos deben ejecutarse concurrentemente y el control debe volver al shell SOLAMENTE cuando ambos hayan acabado.

(b) **La implementación es incorrecta**: Propóngase una implementación correcta.

(c) **El comando *duplo* no es implementable**: Razónese.

El código es correcto. Para hacer que la ejecución de ambos procesos sea en primer plano (ejecutándose ambos concurrentemente y que el control vuelva al shell SOLAMENTE cuando

ambos hayan acabado) no es correcto cambiar las llamadas a *EjecutarProceso* en *cmd_duplo* sustituyendo el segundo prámetro por un 1, ya que esto haría que ambos procesos no se ejecutasen concurrentemente. Una posible solución sería

```
.............
pid_t EjecutarProceso (char * args[],int primerplano) {
  pid_t pid;
  if ((pid=fork())==-1) {
      perror ("Imposible crear proceso"); return;
  }
  if ((pid==0){
      Ejecutar(args);
      perror ("Fallo en Ejecutar");
      exit(255);
  } else if (primerplano)
           waitpid(pid, NULL, 0);
         else
           InsertarListaProcesos (pid,args);
  return pid;
}

void cmd_duplo( char * args[]) {
   pid_t p1,p2;

   p1=EjecutarProceso(args,0);
   p2=EjecutarProceso(args,0);

   waitpid(p1,NULL,0);
   waitpid(p2,NULL,0);
}
```