Apartado	1	2	3	4	5	☐ TGR
Puntuación						

☐ No Presentado

EXAMEN DE SISTEMAS OPERATIVOS (Grado en Ing. Informática) 17/1/2013.

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**) En un sistema de archivos (tipo *system V*) el usuario `juan` tiene su buzón de correo en el archivo `/var/spool/mail/alumnos/bujuan` del que es propietario. Responder a lo siguiente (**Cada respuesta solo se puntuará si es correcta**):

   (a) Calcular el número de accesos a disco necesarios como mínimo en la operación:

   `open("/var/spool/mail/alumnos/bujuan", O_RDONLY)`

   para obtener el i-nodo de `bujuan` si las entradas de los diferentes subdirectorios se encuentran siempre en el primer bloque del directorio padre, excepto `alumnos` que se encuentra en el cuarto y `bujuan` que se encuentra en el tercero. Las cachés de datos e inodos están inicialmente vacías.

   Número mínimo de accesos: [11] =

   Número mínimo de accesos al área de datos: [10] +

   Número mínimo de accesos a la lista de inodos: [1] (0,4 puntos)

   (b) El sistema de archivos tiene un tamaño de bloque de 1 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 4 bytes. Calcula cuántos bloques de disco son necesarios para representar el archivo `bujuan` cuando su tamaño es de 1,5MBytes. Discriminar cuántos bloques son de datos y cuántos de índices.

   Núm. bloques de datos: [1536] Núm. bloques de índices: [7] (0,4 puntos)

   (c) Calcular el número de bloque lógico en el sistema de ficheros correspondiente al inodo del raíz (se comienza a contar en el bloque lógico 0) y el del inodo del fichero `bujuan`, suponiendo lo siguiente:

   - El número de inodo del "/" es el 2, y al fichero `bujuan` le corresponde el inodo núm. 163 (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* 8 bloques.

   Número bloque lógico inodo "/" = [9]

   Número bloque lógico inodo fichero *bujuan* = [19] (0,4 puntos)

   (d) Supongamos la siguiente secuencia de comandos desde el directorio `/var/spool/mail/alumnos/`: se crea un enlace simbólico `slink` al archivo `bujuan`, con el comando `ln -s bujuan slink`, (el comando `ls -l` mostraría `slink -> bujuan`). Contesta a lo siguiente:

   (sólo se puntúa si son correctas las tres respuestas) (0,3 puntos)

   A) Ambos ficheros tiene el mismo número de inodo (Cierto/Falso [Falso])

   B) Si posteriormente se realizan las acciones:

   ```
   ln bujuan bujuan2 /* crea hard link bujuan2 */
   rm bujuan
   ```

   se puede seguir accediendo a través del fichero `slink` al fichero `bujuan2` (Cierto/Falso [Falso])

C) Indica el tamaño en bytes del fichero *slink* [6]

2. (**2 puntos**) Considérese un sistema de memoria virtual con las siguientes características:

   - Tamaño de página=2Kb (2^{11} bytes). Memoria física = 32 Gb (2^{35} bytes).
   - Direcciones virtuales de 32 bits, aunque no se utilizan todos los bits
   - Tablas de páginas en **dos niveles**, implementadas en memoria: cada proceso tiene una tabla de páginas de primer nivel (llamada *page directory*), cada una de cuyas entradas apunta a una tabla de páginas de segundo nivel y cada entrada de la tabla de páginas de segundo nivel apunta a un marco de memoria física.
   - En cada entrada de las tablas de páginas y del *page directory* hay 7 bits utilizados por el sistema de memoria virtual (página presente, página accedida, página modificada página solo lectura ...).

   Contéstese razonando brevemente a las siguientes cuestiones

   a) ¿Cuantos marcos de memoria física hay en dicha arquitectura? [2^{24}] marcos

      $2^{35}/2^{11} = 2^{24}$

   b) ¿Cuál es el número mínimo de bits en una entrada de la tabla de páginas de 2^o nivel? [31] bits

      Se necesitan 24 bits para direccionar los marcos y los 7 bits utilizados por el sistema de memoria virtual

   c) ¿Cual es el número mínimo de bits en una entrada de la tabla de páginas de 1^{er} nivel? [31] bits
      Se necesitan 24 bits para direccionar los marcos ya que las tablas de páginas de segundo nivel pueden residir en cualquier marco de memoria. Recuerde que el enunciado indica que las tablas se implementan en memoria.

   d) Suponiendo que el tamaño de cada entrada de la tabla de páginas se redondea al múltiplo mas cercano de 4 bytes y que cada tabla de páginas debe caber en una página. ¿Cuántos de los 32 bits de direccionamiento se usan *realmente*?
      [29] bits usados realmente

      Cada entrada de la tabla de páginas tiene 32 bits (2^2 bytes) por el padding. Las páginas son de 2Kb (2^{11} bytes). Por tanto hay $2^{11}/2^2 = 2^9 = 512$ entradas en las tablas de páginas y en el directorio de páginas (tabla de primer nivel). Las páginas son de 2Kb, por los que se necesitan 11 bits para especificar el offset. Por tanto para una dirección lóginca se necesitan: 9 bits para indicar la entrada en el directorio de páginas, 9 bits para indicar la entrada en la tabla de segundo nivel, 11 bits para el offset en la página (total 29 bits).

   ¿Qué tamaño tiene el espacio virtual de un proceso? [512 Mb]

   2^{29} bytes = 512 Mb. En este ejercio ocurre que el espacio virtual un proceso es menor que la cantidad de memoria física, pero esto es algo que también ocurre en muchos sistemas con la arquitectura de 32 bits x86 donde el espacio virtual de un proceso son 4 Gb y muchos sistemas tiene más memoria física.

3. (**1,5 puntos**) Para ejecutar programas sin crear proceso y creando proceso en primer y segundo plano, un intérprete de comandos utiliza las funciones que se muestran a continuación. Se supone además que el símbolo @ indica al shell que el programa en cuestion ha de ejecutarse con la prioridad cambiada a la cantidad especificada tras dicho símbolo.

```
int CambiarPri (char * args[], int *ppri) {
   int i;
   for (i=0; args[i]!=NULL; i++)
      if (args[i][0]=='@'){
         *ppri=atoi(args[i]+1);
         args[i]=NULL;  /*uno*/
         return 1;
      }
   return 0;
}

int Ejecutar (char * args[]) {
   int pri;
   if (CambiarPri(args, &pri)){
       if (setpriority(PRIO_PROCESS, getpid(), pri)==-1) {
           perror ("Imposible cambiar prioridad");
           return -1;
       }
    }
   if (args[0]==NULL){
       errno=EFAULT; /*dos*/
       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=fork())!=0){
       Ejecutar(args);
       perror ("Fallo en Ejecutar");
       exit(255);  /*tres*/
   }
   else if (primerplano)
          waitpid(pid, NULL, WNOHANG | WUNTRACED | WCONTINUED);
        else
          InsertarListaProcesos (pid,args);
}
```

Conteste **CATEGÓRICAMENTE** si esta implementación es:

☐ **INCORRECTA**: Dígase por qué y corríjase sobre el código.

☐ **CORRECTA**: Respóndase a las siguientes cuestiones (a)-(e):

(a) ¿Es necesaria la sentencia marcada con /*uno*/? ¿por qué?

(b) ¿Es necesaria la sentencia marcada con /*tres*/? ¿por qué?

(c) Si se intenta ejecutar en segundo plano un programa que no existe ¿Como aparecerá en la lista de procesos en segundo plano? (corriendo, terminado por señal, terminado normalmente...)

(d) ¿Para que sirve la sentencia /*dos*/?

(e) Modifíquese el código para que el programa sea ejecutado aun en el caso de no poder cambiar la prioridad.

Use el espacio de debajo en cualquiera de los dos casos (correcta o incorrecta).

LA SOLUCION ES INCORRECTA

- hay dos llamadas a *fork()* que crean TRES procesos
- *fork()* devuelve 0 al proceso hijo y el pid del proceso hijo al proceso padre, con lo que el programa sería ejecutado por el padre y la lista de procesos en segundo plano no se rellenaría
- la opción de WNOHANG de *waitpid()* hace que *waitpid()* no espere, y por tanto no habría ejecución en primer plano

El código corregido podría quedar así:

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

4. (**1 punto**) Supóngase que tenemos un disco duro con 250 cilindros numerados del 0 al 249, cuya cabeza está sobre el cilindro 100 (**atendiendo una petición de acceso previa**) y se está moviendo en sentido **ascendente** ($0 \rightarrow 249$). Si en el instante actual la cola de peticiones de acceso a cilindros contiene las peticiones: $\langle 99, 102, 125, 29, 247, 95, 110, 90, 198, 10 \rangle$, indíquese en qué orden se atenderán dichas peticiones si se utiliza un algoritmo de planificación:

FCFS o FIFO:	99	102	125	29	247	95	110	90	198	10
SSTF:	99	102	95	90	110	125	198	247	29	10
SCAN:	102	110	125	198	247	99	95	90	29	10
C-SCAN:	102	110	125	198	247	10	29	90	95	98

5. (**1 punto**) A la hora de fijar el quantum q en el algoritmo de planificación *Round Robin*:

   (a) ¿Qué problema puede haber si q es demasiado pequeño respecto al tamaño de las ráfagas de CPU? Un q demasiado pequeño hace que se vuelva significativo el tiempo empleado en cambios de contexto (mucho más frecuentes) respecto al empleado en ejecución CPU de los procesos.

   Respuesta **incorrecta** más frecuente: "*un q pequeño aumenta la lista de procesos en espera*". Esta respuesta no es correcta: si tenemos 10 procesos que desean usar una única CPU simultáneamente, siempre habrá uno en CPU y 9 en espera, independientemente de si se producen más o menos cambios de contexto entre ellos por culpa del factor q. De hecho, es más bien al contrario: al aumentar q (y no al disminuirlo) podemos aumentar el tamaño medio de la lista de espera, ya que un proceso con una ráfaga larga hace que procesos cortos se vayan metiendo en espera a lo largo de la duración del cuanto.

   (b) Por el contrario, ¿qué sucede cuando q es muy grande, mayor incluso que la ráfaga CPU más larga? El algoritmo deja de ser apropiativo y pasa a convertirse en un FCFS (First-Come First-Served) con el consiguiente problema de efecto "convoy" (1 proceso con ráfagas de CPU largas bloquea a muchos procesos con ráfagas de CPU cortas.

   Respuesta **incorrecta** más frecuente: "*se desperdicia tiempo de CPU porque después de acabar la ráfaga, espera parada a que acabe el cuanto*". Esta respuesta es incorrecta: en ningún algoritmo de planificación un proceso hace esperar a la CPU después de acabar su ráfaga. Cuando eso sucede, la CPU queda libre inmediatamente y se pasa a tomar otro proceso de la lista de procesos en espera. Nótese que si esperase a acabar el cuanto, sería **peor** que FCFS.