

|            |   |   |   |   |   |   |   |                              |
|------------|---|---|---|---|---|---|---|------------------------------|
| Apartado   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | <input type="checkbox"/> TGR |
| Puntuación |   |   |   |   |   |   |   |                              |

☐ No Presentado

## EXAMEN DE SISTEMAS OPERATIVOS (Grado en Ing. Informática) 20/1/2014.

**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 10 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 8 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 8 MBytes. Discriminar cuántos bloques son de datos y cuántos de índices.

Número de bloques de datos:

Número de bloques de índices:

(0.4 puntos)

- (b) Un proceso abre ese fichero:

```
fd=open("/home/usr1/direcciones", O_RDONLY)
```

y, una vez abierto el archivo, lo primero que realiza el proceso es `lseek(fd, 15006, SEEK_SET)`. Calcula cuántos bloques habría que leer de disco para la operación de lectura de un carácter: `c=fgetc(fd)`, suponiendo el BC vacío. (`SEEK_SET` indica que el desplazamiento se considera a partir del origen del fichero).

Número de bloques a leer =

Misma pregunta, pero considerando que lo primero que realiza el proceso es:

```
lseek(fd, 6291456, SEEK_SET) (Nota:  $2097152 = 6 * 2^{20}$ ).
```

Número de bloques a leer =

(0.3 puntos)

- (c) En ese sistema de archivos UNIX con un tamaño de inodos de 64 bytes, tamaño de bloque de 2Kbytes, la zona de inodos (lista de inodos) ocupa 2048 bloques. El superbloque mantiene un mapa de bits de inodos libres para determinar los inodos libres/ocupados de la lista de inodos. Calcular cuántos bloques ocupa ese mapa de bits de inodos libres.

Número bloques del mapa de bits =

(0.4 puntos)

- (c) Supongamos la siguiente secuencia de comandos desde el directorio `/home/usr1`: se crea un enlace simbólico `slink` al archivo `direcciones` (cuyo número de inodo es 120345), con el comando `ln -s direcciones slink`, (el comando `ls -l` mostraría `slink → direcciones`).

Contestar a lo siguiente:

(0.4 puntos)

A. Indicar el tamaño del fichero `slink`

B. El inodo del fichero `slink` es el 120345 (Cierto/Falso )

C. Si posteriormente se realizan las acciones:

```
ln direcciones direcciones2 /* crea hard link direcciones2 */
rm direcciones
```

se puede seguir accediendo a través del fichero `slink` al fichero `direcciones2` (Cierto/Falso

)

D. La operación anterior de borrado de datos en disco puede provocar errores de consistencia si hay una caída de alimentación durante la operación. Si el sistema de ficheros Unix es basado en registro (journalist file system), los posibles errores de cambio o movimiento de datos o metadatos en el disco se eliminan por completo. Indicar si es cierto/falso esta última afirmación. (Cierto/Falso )

## TABLA PARA EJERCICIO 2

DADOS PROBLEMA MEMORIA

### Virtual Address Format

| Virtual seg #<br>(4 bits) | Virtual Page #<br>(8 bits) | Offset<br>(8 bits) |
|---------------------------|----------------------------|--------------------|
|---------------------------|----------------------------|--------------------|

### Segment Table (Max Segment=3)

| Seg # | Page Table Base | Max Page Entries | Segment State |
|-------|-----------------|------------------|---------------|
| 0     | 0x2030          | 0x20             | Valid         |
| 1     | 0x1020          | 0x10             | Valid         |
| 2     | 0x3110          | 0x40             | Invalid       |
| 3     | 0x4000          | 0x20             | Valid         |

### Page Table Entry

| First Byte           | Second Byte   |
|----------------------|---|
| Physical Page Number | 0x00 = Invalid<br>0x06 = Valid, RO<br>0x07 = Valid, R/W |

### Physical Memory

| Address | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 | +9 | +A | +B | +C | +D | +E | +F |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0x0000  | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| 0x0010  | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D |
| ....    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x1010  | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
| 0x1020  | 40 | 07 | 41 | 06 | 30 | 06 | 31 | 07 | 00 | 07 | 00 | 00 | 00 | 00 | 00 | 00 |
| ....    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x2000  | 02 | 20 | 03 | 30 | 04 | 40 | 05 | 50 | 06 | 60 | 07 | 70 | 08 | 80 | 09 | 90 |
| 0x2010  | 0A | A0 | 0B | B0 | 0C | C0 | 0D | D0 | 0E | E0 | 0F | F0 | 10 | 01 | 11 | 11 |
| 0x2020  | 12 | 21 | 13 | 31 | 14 | 41 | 15 | 51 | 16 | 61 | 17 | 71 | 18 | 81 | 19 | 91 |
| 0x2030  | 10 | 06 | 11 | 00 | 12 | 07 | 40 | 07 | 41 | 07 | 00 | 00 | 00 | 00 | 00 | 00 |
| ....    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x30F0  | 00 | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 | AA | BB | CC | DD | EE | FF |
| 0x3100  | 01 | 12 | 23 | 34 | 45 | 56 | 67 | 78 | 89 | 9A | AB | BC | CD | DE | EF | 00 |
| 0x3110  | 02 | 13 | 24 | 35 | 46 | 57 | 68 | 79 | 8A | 9B | AC | BD | CE | DF | F0 | 01 |
| 0x3120  | 03 | 06 | 25 | 36 | 47 | 58 | 69 | 7A | 8B | 9C | AD | BE | CF | E0 | F1 | 02 |
| 0x3130  | 04 | 15 | 26 | 37 | 48 | 59 | 70 | 7B | 8C | 9D | AE | BF | D0 | E1 | F2 | 03 |
| ....    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x4000  | 30 | 00 | 31 | 06 | 32 | 07 | 33 | 07 | 34 | 06 | 35 | 00 | 43 | 38 | 32 | 79 |
| 0x4010  | 50 | 28 | 84 | 19 | 71 | 69 | 39 | 93 | 75 | 10 | 58 | 20 | 97 | 49 | 44 | 59 |
| 0x4020  | 23 | 07 | 20 | 07 | 00 | 06 | 62 | 08 | 99 | 86 | 28 | 03 | 48 | 25 | 34 | 21 |

2. (1 punto) Un sistema utiliza segmentación paginada. En la hoja adjunta se ofrecen los formatos de las direcciones virtuales, la tabla de segmentos, el formato de las entradas de la tabla de páginas, y los contenidos de la memoria física necesarios para hacer la traslación de direcciones virtuales a direcciones físicas que es el objeto de este ejercicio. Las entradas de la tabla de segmentos apuntan a tablas de páginas en memoria. Una tabla de páginas consta de una serie de entradas de 16 bits (PTEs, *Page Table Entries*). El primer byte de cada PTE es un número de página física de 8 bits, y el segundo byte contiene uno de los valores que se indican en la tabla de la hoja adjunta. En la tabla de traslación de direcciones, en el caso de un error, se debe indicar de que tipo es: **error de segmento**: segmento inválido o no definido, **segment overflow**: dirección fuera del rango del segmento, **error de acceso**: página inválida, o intento de escribir una página de sólo lectura.

| ****Direcciones Virtuales**** | ****Direcciones Físicas**** |
|-------------------------------|-----------------------------|
| 0x10123                       | 0x4123                      |
| 0x33423                       | segment overflow            |
| 0x20456                       | error de segmento           |
| 0x31056                       | 0x2356                      |
| 0x10400                       | 0x0000                      |
| 0x00278                       | 0x1278                      |

3. (1 punto) Para cada una de las siguientes cuestiones, enmarcar la respuesta correcta (Verdadero/Falso) y justificar las respuestas. Marcas ambiguas o justificaciones insuficientes o erróneas invalidan la cuestión.

- Verdadero / Falso. En un sistema con memoria virtual y paginación, tabla de páginas (TP) de un nivel y TLB, una referencia a memoria puede producir un fallo en la TLB y no fallo en la TP.  
Verdadero. Es debido a que una TLB normalmente no puede contener todas las entradas de la TP de un proceso.
- Verdadero / Falso. En un sistema con memoria virtual y paginación, tabla de páginas (TP) de un nivel y TLB, una referencia a memoria puede producir un fallo en la TLB y un fallo en la TP.  
Verdadero. Es el caso de un fallo de página.
- Verdadero / Falso. En un sistema con memoria virtual y paginación y tabla de páginas (TP) de un nivel, el número de entradas en la TP, y por tanto el tamaño de la TP, viene dado por el número de páginas que están en memoria física.  
Falso. Viene dado por el número de páginas lógicas del proceso.
- Verdadero / Falso. El algoritmo PFF (Frecuencia de Fallo de Página) necesita para su implementación registrar el tiempo en el que se producen todas las referencias a memoria.  
Falso. Es suficiente, y así se implementa, un contador de referencias a páginas para registrar la frecuencia de fallo.

4. (0,5 puntos) En un sistema unix un proceso del *root*

- a) Se ejecuta siempre en modo kernel
  - b) Se ejecuta siempre en modo usuario
  - c) El *root* puede decidir que porcentaje debe ejecutarse en modo kernel mediante el comando *time*
  - d) El *root* puede configurar que porcentaje del tiempo de CPU se ejecutan sus procesos (y los de otros usuarios) en modo kernel, pero una vez configurado este porcentaje no puede cambiarse
  - e) Depende de la prioridad del proceso, a mayores prioridades mayor tiempo en modo kernel
  - f) Depende de la prioridad del proceso, a mayores prioridades menor tiempo en modo kernel
  - g) Ninguna de las anteriores
5. (1 punto) Un intérprete de comandos mantiene una lista de procesos en segundo plano, cada proceso está descrito por una estructura como la que se ve a continuación, donde el campo *val* se utiliza para guardar el valor devuelto por el proceso cuando termina, o el número de la señal que lo ha terminado o parado, si ha terminado o parado debido a una señal; este campo tiene un valor indefinido si el proceso está activo

```
typedef struct PROCESO {
    pid_t pid;
    time_t fecha;
    char linea[MAXLINEA];
    int pri;
    int estado;
    int val; /*guarda la senal o el valor devuelto*/
} proceso_t;
```

Para actualizar el estado de los procesos el shell utiliza una función como la siguiente:

```
void ActualizarProceso (proceso_t *p)
{
    int valor=0;
    int estado;

    ObtenerPrioridad (p->pid,&p->pri);
    if ((estado=ObtenerEstadoProceso (p->pid, &valor))!=-1){
        p->estado=estado;
        if (p->estado!=ACTIVO)
            p->val=valor;
    }
}

int ObtenerEstadoProceso (pid_t pid, int * val){
    int status;
    int flags=WNOHANG|WUNTRACED|WCONTINUED;

    if (waitpid(pid,&status, flags)==pid){
        if (WIFEXITED (status))
            {*val=WEXITSTATUS(status); return EXITED;}
    }
```

```

        else if (WIFSIGNALED(status))
            {*val= WTERMSIG(status); return SIGNALED;}
        else if (WIFSTOPPED(status))
            {*val= WSTOPSIG(status); return STOPPED;}
        else if WIFCONTINUED(status)
            return ACTIVO;
    }
    else return -1;
}

```

Suponiendo que la función *ObtenerPrioridad* funciona correctamente, dígame CATEGÓRICAMENTE si es correcta o incorrecta la implementación mostrada

- Es correcta. Explíquese cual sería el funcionamiento si se eliminasen los flags WUNTRACED y WCONTINUED.
- Es incorrecta. Corríjase

a) La solución es correcta.

Al no tener esos flags, *waitpid* no informará de los procesos que paran (debido a una señal) ni de los que continúan (WCONTINUED). Por tanto el funcionamiento del shell sería correcto salvo que los procesos en segundo plano solo los consideraría “activos”, “terminados normalmente” o “terminados debido a una señal”.

- (1 punto) Suponiendo que el fichero “input.txt” contiene los siguientes 13 bytes: “EXAMEN DE SO\n”, y que el siguiente código no produce ningún error de ejecución:

```

#include "includes.h"
int main(){
    int fd = open("fichero.txt",O_RDONLY);
    char BUF[5]= {'-','-','-','-','\0'};

```

|  |  |  |
|--|--|--|
| <pre> #ifdef CASO_A     read(fd,BUF,1);     read(fd,BUF,3);  #endif </pre> | <pre> #ifdef CASO_B     pread(fd,BUF,1,0);     pread(fd,BUF,3,0);  #endif </pre> | <pre> #ifdef CASO_C     struct iovec iov[2];     iov[0].iov_base= BUF;     iov[0].iov_len= 1;     iov[1].iov_base= BUF+1;     iov[1].iov_len= 3;     readv(fd,iov,2);  #endif </pre> |
|--|--|--|

```

printf("%s",BUF);
close(fd);
}

```

Muéstrese el contenido del buffer **BUF** ( $BUF[i], i = 0 \dots 4$ ) en función de si la lectura se hace con: read, pread, o con readv.

|            |              |   |   |   |    |    |               |   |   |   |    |    |               |   |   |   |    |
|------------|--------------|---|---|---|----|----|---------------|---|---|---|----|----|---------------|---|---|---|----|
|            | CASO_A: read |   |   |   |    |    | CASO_B: pread |   |   |   |    |    | CASO_C: readv |   |   |   |    |
|            | X            | A | M | - | \0 |    | E             | X | A | - | \0 |    | E             | X | A | M | \0 |
| BUF[i], i= | 0            | 1 | 2 | 3 | 4  | i= | 0             | 1 | 2 | 3 | 4  | i= | 0             | 1 | 2 | 3 | 4  |

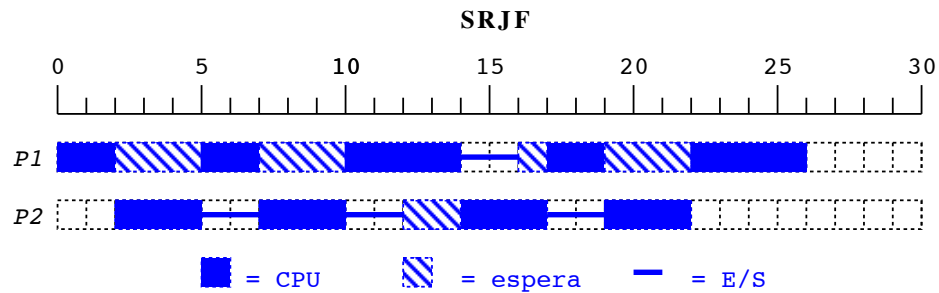
**Nota:** Recuerdese que los prototipos de las funciones read, readp y readv son los siguientes:

```

ssize_t read (int fd, void *buf, size_t count);
ssize_t pread(int fd, void *buf, size_t count, off_t offset);
ssize_t readv(int fd, const struct iovec *iov, int iovcnt);

```

7. (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  $P1 = (\underline{8}, 2, \underline{6})$ ,  $P2 = (\underline{3}, 2, \underline{3}, 2, \underline{3}, 2, \underline{3})$  donde los números subrayados representan tiempo de CPU y el resto es E/S. Dibuja el diagrama de ejecución para el algoritmo Shortest Remaining Time First (SRTF). Calcula el **tiempo de retorno** para cada proceso.



Tiempo de retorno  $P1 =$  26  $ms$

Tiempo de retorno  $P2 =$  22-2 = 20  $ms$