SISTEMAS OPERATIVOS - CUESTIONES
8 de Febrero de 2017

Nombre _____ DNI _____

Apellidos _____ Grupo _____

Cuestión 1. (1 punto) Un usuario ejecuta los siguientes comandos en un SO con sistema de ficheros tipo UNIX:

$ mkdir usr
$ cd home/carpeta/
$ ln nada.odt cosas.odt
$ ln –s ../archivo.txt acceso
$ rm nada.odt

Tras ejecutar los comandos, las estructuras del sistema de ficheros almacenan la siguiente información:

Mapa de bits (el bit de más a la izquierda representa el bloque 1): 1111110111000010

Tabla de nodos-i

nodo-i	2	4	7	8	9	10	11	16
Tipo	D	D	D	F	E	F	F	D
Enlaces	NA	NA	NA	1	1	1	1	NA
Directo	1	2	4	5	10	15	9	3
Directo	null	null	null	6	null	8	null	null

Contenido de bloques relevantes (los bloques que no figuran aquí contienen datos o están vacíos)

Bloque 1

.	2
..	2
home	4
usr	7

Bloque 2

.	4
..	2
carpeta	16
archivo.txt	11

Bloque 3

.	16
..	4
valores.odt	8
acceso	9
cosas.odt	10

Bloque 4

.	7
..	2

Indicar el contenido de la tabla de nodos-i, el contenido de los bloques relevantes y el mapa de bits antes de la ejecución de los comandos citados anteriormente.

Cuestión 2. (1 punto) Considere el siguiente fragmento de código de un módulo del kernel Linux.

```
static int device_open(struct inode *ino, struct file
*f) {... }

static int device_release(struct inode *ino, struct
file *f) {...}

static ssize_t device_read(struct file *ino, char *
buf, size_t len, loff_t *off) {...}

#define DEVICE_NAME “foo”

static struct file_operations fops = {
        .read = device_read,
        .open = device_open,
        .release = device_release
};
```

```
int init_module(void) {
  int major;
  int minor;
  if (alloc_chrdev_region(&start,0, 1,DEVICE_NAME)) {
        printk(KERN_INFO "Can't allocate chrdev_region()");
        return -ENOMEM;
  }

  if ((chardev=cdev_alloc())==NULL) {
        printk(KERN_INFO "cdev_alloc() failed ");
        return -ENOMEM;
  }

  cdev_init(chardev,&fops);
  if (cdev_add(chardev,start,1)) {
        printk(KERN_INFO "cdev_add() failed ");
        return -ENOMEM;
  }

  major=MAJOR(start);
  minor=MINOR(start);
  return SUCCESS;
}
```

Explique qué hacen las funciones alloc_chrdev_region(), cdev_alloc(), cdev_init() y cdev_add(), en el contexto de la función de inicialización del módulo. Indique también el significado de los parámetros de cada función.

Cuestión 3. (1 punto) Se dispone de un conjunto de tareas (de T1 a T4) cuyos patrones de ejecución están indicados en la siguiente tabla

Tarea	Llegada	CPU	E/S	CPU	E/S	CPU
T1	0	2	2	2	2	1
T2	0	8				
T3	2	2	4	1		
T4	5	6	2	1		

Complete el diagrama de planificación que se muestra a continuación suponiendo que se aplica el algoritmo Round Robin con q=3 y el sistema está dotado de una única CPU. Indicar también el tiempo de espera de cada proceso, la productividad y el porcentaje de uso de CPU.

A la hora de rellenar el diagrama, se ha de usar el siguiente convenio para representar los estados de las tareas:

- En ejecución: marcar con "**X**"
- Bloqueado por E/S: marcar con **"--"**
- Listo para ejecutar: marcar con **"O"**

Nota: Si hay alguna situación de conflicto (dos tareas se insertan en la cola en el mismo instante) tendrá preferencia la tarea de menor número (p.ej. T1 antes que T2, T3 antes que T4,...)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
T1																																
T2																																
T3																																
T4																																

Cuestión 4. (1 punto) Considere la siguiente solución al problema de los lectores/escritores vista en clase que hace uso de dos semáforos (sem_nreaders, sem_read_write) y un contador (nr_readers), para controlar el acceso a una variable entera compartida (data):

```
/* Código del hilo lector */
void Reader(void) {
  while(1){
        sem_wait(&sem_nreaders);
        nr_readers = nr_readers + 1;
        if (nr_readers == 1)
          sem_wait(&sem_read_write);
        sem_post(&sem_nreaders);

        /* read data */
        printf("%d\n", data);

        sem_wait(&sem_nreaders);
        nr_readers = nr_readers - 1;
        if (nr_readers == 0)
            sem_post(&sem_read_write);
        sem_post(&sem_nreaders);
  }
}
```

```
/* Código del hilo escritor */
void Writer(void) {
  while(1){
    sem_wait(&sem_read_write);

    /* modify the resource */
     data = data + 2;

     sem_post(&sem_read_write);

  }
}
```

a. Suponga que la variable nr_readers está inicializada a 0 y los contadores de los semáforos se inicializan a 1. ¿Con la solución proporcionada, tiene alguno de los dos colectivos (lectores o escritores) más prioridad que el otro a la hora de acceder a la sección crítica? Justifique su respuesta.

b. En el código proporcionado los semáforos se utilizan para garantizar exclusión mutua. ¿Dado que los cerrojos o mutexes se utilizan también para imponer exclusión mutua, podría reemplazarse alguno de los semáforos (o ambos) por un cerrojo o mutex ? Indique qué semáforo(s) podría(n) reemplazarse y cuál(es) no, indicando el motivo. En caso de que fuera posible reemplazar alguno de los semáforos por un mutex, indique también qué modificaciones deberían realizarse en el código anterior para que funcionara correctamente.

Cuestión 5. (1 punto) Un proceso en UNIX ejecuta el siguiente código:

```
int main(void) {
  int child_status;
  int numB=0,var=0, pid=0;
  char c;
  int fd1, fd2;

  fd1=open("text.txt", O_RDWR);

  pid = fork();

  if (pid == 0) {
    int j;
    char buf[1];

    fd2=open("text.txt", O_RDONLY);

    for (j = 0; j < 14; j++)
         var=var+1;

    while((c=read(fd2,buf, 1))!=0)
        numB = numB+1;
label_child:
    close(fd2);
    exit(0);
 }
 else if (pid > 0){
        int j;
        char buf[1];

        for (j = 0; j < 6; j++)
        var=var+1;

        while((c=read(fd1,buf,1))!=0)
           numB = numB+1;
label_parent:
        wait(&child_status);

     close(fd1);
        exit(0);
  } else {
        fprintf(stderr, "can't fork,
       error %d\n", errno);
        exit(EXIT_FAILURE);
  }
  return 0;
}
```

Conteste las siguientes preguntas:

a) ¿Qué valor tiene la variable var del padre y la del hijo inmediatamente después de ejecutarse el fork()?

b) Completar el contenido de las tablas que se muestran a continuación considerando que la ejecución de los procesos se encuentra en el siguiente punto: los procesos padre e hijo ejecutarán a continuación la sentencia que se encuentra tras las etiquetas `label_parent` y `label_child`, respectivamente.

TDDA[1] Padre

DF	Ind. TFA
0	230
1	564
2	28
3	12
4	

TDDA Hijo

DF	Ind. TFA
0	230
1	564
2	28
3	
4	

Tabla intermedia de posiciones (TFA)

	Pos L/E	Num nodo-i	Perm.	Cont. refs
...	...	...	...	...
12		57		
13				
14	...	...	...	...
...	...	...	...	...

Tabla nodos-i

Nodo-i	Cont.
….	…..
Nodo-i 57	
...	….

c) ¿Qué valor tiene la variable numB del padre y del hijo después de ejecutarse el padre?

1Tabla de descriptores de ficheros (numéricos) abiertos por el proceso

Cuestión 6. (1 punto) Considerar un sistema en el que utiliza gestión de memoria virtual con paginación bajo demanda con páginas de 1KB, direcciones de memoria de 64 bits, memoria principal de 16G. Se quiere ejecutar en dicho sistema el siguiente programa.

```
#define INFILE "./indata.dat"
#define OUTFILE "./outdata.dat"
#define FILE_SIZE 2048

void process(char *inbuf, char *outbuf, int size);
char checksum(char *inbuf, int size);

char table[]="abcdefghijklmnopqrstuvwxyz";
char *buf;

int main(int argc, char* argv[])
{
  pid_t pid;
  char *proj;
  char output;
  int fdi, fdo;
  int status;
  char infile[]=INFILE;
  char outfile[]=OUTFILE;

  //A
  buf= (char *) malloc(FILE_SIZE*sizeof(char));

  if((fdi=open(infile,O_RDONLY))<0) {
    fprintf(stderr,"open error%d\n", errno);
    exit(EXIT_FAILURE);
  }

  if(read(fdi,buf,FILE_SIZE)==-1) {
    fprintf(stderr,"read error%d\n", errno);
    exit(EXIT_FAILURE);
  }

 //B
  if((pid = fork())== -1) {
    fprintf(stderr,"fork error%d\n", errno);
    exit(EXIT_FAILURE);
  }
```

```
  else if(pid != 0){
        fdo = open(outfile,O_RDWR|O_CREAT|
O_TRUNC,
                        (mode_t)0600);
        proj=(char *) malloc(FILE_SIZE*
                sizeof(char));
        //C
        process(buf,proj,FILE_SIZE);
        if(write(fdo,proj,FILE_SIZE)==-1){
        fprintf(stderr,"write error%d\n",errno);
        exit(EXIT_FAILURE);
     }
     close(fdo);
        wait(&status);
   } else {
        output=checksum(buf,FILE_SIZE);
        printf("checksum = %x\n",output & 0xff);
        close(fdi);
        exit(0);
  }
  close(fdi);
  exit(0);
}

void process(char *inbuf, char *outbuf, int size){
  int i;
  for(i=0;i<size;i++) {
        if(inbuf[i]>='a' && inbuf[i]<='z')
        outbuf[i]=inbuf[i];
        else
        outbuf[i]=table[i%25];
  }
}

char checksum(char *inbuf, int size) {
  char sum=0;
  for(int i=0;i<size;i++)
        sum ^= inbuf[i];
  return(sum);
}
```

Considerando que:
- Hay suficientes marcos de página libres en el sistema.
- El contenido del fichero “indata.dat” ocupa 2KB.
- El texto (código) del programa tras la compilación ocupa 4 páginas.
- El contenido inicial de la pila al lanzar a ejecución el programa ocupa tan solo 32 bytes.

a) Identificar qué regiones lógicas (y su correspondiente tamaño en páginas) constituyen la imagen de memoria del proceso o procesos que se crean al lanzar a ejecución dicho programa en los puntos marcados en el código como A, B y C. Asumir que las funciones que se usan pero no se definen el programa se encuentran incluidas en la biblioteca estándar de C (libc) y que al generar el ejecutable se ha enlazado la libc de forma estática, ocupando el código(texto) de todas las funciones de la libc 1 página.

b) ¿ Si la libc se hubiera enlazado de forma dinámica, cambiará la respuesta para los puntos marcados como A y B?. Razonar la respuesta.

NOTA: Contestar a esta cuestión en el recuadro que aparece al final del enunciado, tras el problema 2.

SISTEMAS OPERATIVOS - PROBLEMAS
8 de Febrero de 2017

Nombre _____ DNI _____

Apellidos _____ Grupo _____

Problema 1. (2 puntos) Un sistema de ficheros de un SO diseñado a partir de UNIX utiliza bloques de disco de 4 Kbytes de capacidad. Para el direccionamiento de estos bloques se utilizan punteros de 64 bits. Para indicar el tamaño del fichero y el desplazamiento (offset) de la posición en bytes en las operaciones read y write, se utilizan números de 64 bits. Cada nodo-i tiene 10 punteros de direccionamiento directo, 1 puntero indirecto simple y 1 puntero indirecto doble.

En este sistema se quiere programar una aplicación en C que realice la siguiente secuencia de operaciones:
1. Creación de un fichero de nombre “outfile” en el directorio de trabajo actual. Si el fichero ya existe ser truncará su contenido.
2. Escritura de 11 bloques consecutivos en dicho fichero “outfile”, que etiquetaremos para el propósito de este problema como Bloque 0 a Bloque 10.
3. Posicionamiento al comienzo del Bloque 10 y Lectura completa del Bloque 10
4. Posicionamiento al comienzo del fichero y Lectura completa de los 10 primeros bloques del fichero.
5. Posicionamiento en el byte 3051270. Escritura de la cadena “fin de fichero”.

Se pide:

a. Escribir el programa descrito indicando claramente las llamadas al sistema de ficheros.
b. Indique cuántos bloques de datos e índices ocupa el fichero “outfile” al finalizar la ejecución del programa, y determine el nivel de punteros en el que se encuentra el byte 3051270 del fichero.
c. ¿Cuál es el tamaño máximo de un fichero en este sistema suponiendo despreciable el espacio ocupado por el superbloque y la tabla de nodos-i? Justifique la respuesta.

Problema 2. (2 puntos) En un gran premio de fórmula 1 participan N equipos, con 1 coche por equipo. Cada equipo tiene asociado un par de mecánicos, que tienen a su disposición un surtidor de combustible de M litros de capacidad instalado en el box del equipo. A su vez, cada equipo dispone de un camión cisterna situado fuera del box, con un almacén mayor de combustible, suficiente para toda la carrera.

El comportamiento de los dos mecánicos de cada equipo (hilos de un programa concurrente) viene dado por el siguiente código (aún inseguro):

```
void mecánico1(int equipo){
  while(true){
      esperar_llegada_coche(equipo);
      llenar_depósito_coche(L,equipo);
   }
}
```

```
void mecánico2(int equipo){
  while(true){
      llenar_depósito_surtidor(M,equipo);
   }
}
```

Cuando un coche entra en boxes (la función esperar_llegada_coche() retorna), el primer mecánico del equipo correspondiente se ocupa de llenar el depósito del coche, que tiene una capacidad de L litros (donde L<<M), usando el surtidor del box. En el caso de que el surtidor no tenga suficiente combustible para el llenado del depósito del coche, el primer mecánico despertará al segundo, que se encargará de reponer el surtidor del equipo, transfiriendo M litros de combustible desde el camión. Una vez que vuelva a haber combustible en el surtidor, el primer mecánico procederá con el llenado del depósito del coche.

Considere que la función esperar_llegada_coche(equipo) --que bloquea al mecánico en cuestión hasta que llegue el coche de su equipo al box-- está ya implementada, y también lo están las funciones llenar_depósito_coche(litros,id_equipo) y llenar_depósito_surtidor(litros,id_equipo), que emulan las acciones de llenado del depósito del coche y del surtidor respectivamente.

Las restricciones de sincronización a imponer en el programa concurrente son las siguientes:

- El mecánico responsable de llenado del depósito del coche no puede invocar llenar_depósito_coche() si no hay suficientes litros en el surtidor del equipo.
- El responsable de llenado del surtidor sólo puede invocar llenar_depósito_surtidor() si el depósito del surtidor está vacío. (Por simplicidad, supóngase que M es múltiplo de L)

Realice las modificaciones necesarias en el código de los mecánicos para garantizar la correcta sincronización utilizando mutexes, variables condicionales y otras variables compartidas. **Nota:** Además de proporcionar el código de la solución propuesta, se ha de describir claramente para qué sirve cada variable/recurso de sincronización utilizado, así como su valor inicial.