**SISTEMAS OPERATIVOS - CUESTIONES**
**14 de enero de 2019**

Nombre _____ DNI _____

Apellidos _____ Grupo _____

**Cuestión 1. (1 punto)**

Un módulo del kernel define las siguientes funciones asociadas a la apertura, cierre, lectura y escritura en sus ficheros de dispositivo asociados, respectivamente:

```
static int Device_Open = 0;
static int counter = 0;
static int device_open(struct inode *inode,
                       struct file *file) {
    if (Device_Open) return -EBUSY;
    Device_Open++;
    counter = counter + 1;
    printk(KERN_ALERT "Cuenta: %d\n", counter);
    try_module_get(THIS_MODULE);
    return SUCCESS;
}
static int device_release(struct inode *inode, struct
file *file) {
    Device_Open--;
    module_put(THIS_MODULE);
    return 0;
}
```

```
static ssize_t
device_read(struct file *filp,
            char *buffer, size_t length,
            loff_t * offset) {
    printk(KERN_ALERT "No soportada.\n");
    return -EPERM;
}

static ssize_t
device_write(struct file *filp,
             const char *buff,
             size_t len, loff_t * off) {
    printk(KERN_ALERT "No soportada.\n");
    return -EPERM;
}
```

Tras la carga del módulo, le es asignado un major number 235, con un rango de dos minor numbers consecutivos (0 y 1). En dicho sistema, la ejecución de la orden `ls -l /dev/ | grep 235` devuelve la siguiente salida:

```
$ ls -lt /dev/ | grep 235
crwxrwxrwx   1 root root      235,   0 ene 10 10:40 dispositivo1
crw-rw-rw-   1 root root      235,   1 ene 10 10:40 dispositivo2
```

Conteste razonadamente a las siguientes preguntas:

1. ¿Qué órdenes se han ejecutado para crear los dos ficheros de dispositivo listados anteriormente?

2. Suponiendo que el anterior módulo ha sido cargado con éxito, (a) ¿qué mensajes aparecerán por pantalla tras la ejecución de cada una de las siguientes órdenes?;

```
cat /dev/dispositivo1
echo $?
echo "1" > /dev/dispositivo2
echo $?
```

(b) ¿qué mensajes aparecerán en el log del sistema tras la ejecución de cada una de las siguientes órdenes?

**Cuestión 2. (1 punto)**

Un sistema utiliza gestión de memoria virtual con paginación bajo demanda con páginas de 1KB, direcciones de memoria de 64 bits y memoria principal de 16G. Se quiere ejecutar en dicho sistema el siguiente programa:

```
#define infile "./Matrixdata.dat"
#define N 10
#define M 2048
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <pthread.h>
int * matriz;
int * partial_sum;

/* Suma los elementos de una fila de
 * la matriz y almacena el resultado
en partial_sum. El argumento que se le
pasa es un puntero a la fila. */

void* suma_pthread(void* arg) {
    int i;
    int sum=0;
    int fila = (*(int *) arg);
    int *vector = &(matriz[fila*M]);
    for (i=0;i<M;i++)
       sum +=vector[i];
    partial_sum[fila] = sum;
    pthread_exit(0);
}
```

```
int main() {
    int fd, i, suma=0;
    pthread_t thread[N];
    if((fd=open(infile,O_RDONLY))<0) {
        perror("Can't open the input file");
        exit(1);
    }
// Punto A
    matriz = (int *) malloc(N * M * sizeof(int));
    i = read(fd, matriz, N * M * sizeof(int));
    if (i< N * m * sizeof(int)) {
       free(matriz);
       perror("Can't read the input file");
       exit (1);
    }
    partial_sum=(int *) malloc(N*sizeof(int));
    for (i=0; i<N;i++)
       pthread_create(&thread[i],NULL,suma_pthread,(void *)&i);

// Punto B
    for (i=0;i<N;i++) {
       pthread_join(thread[i], NULL);
       suma+=partial_sum[i];
    }
    free(matriz);
    free(partial_sum);
    return suma;
}
```

Considerando que:
- Hay suficientes marcos de página libres en el sistema.
- El texto (código) del programa tras la compilación ocupa 1,5KB.
- Las bibliotecas son estáticas y ocupan 7,2KB.
- El contenido inicial de la pila al lanzar a ejecución el programa ocupa tan solo 32 bytes.
- Un entero en C ocupa 4 bytes.

Identifique qué regiones lógicas (y su 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 y B.

| Punto A | Punto B |
|---|---|
| | |

**Cuestión 3. (1 punto)**

En un sistema tipo UNIX, se ejecuta el siguiente programa, que pretende determinar el número de enteros pares presentes en un vector utilizando dos procesos concurrentes:

```
#include <stdio.h>
#include <limits.h>

/* read_vector lee un vector de enteros de un
 * fichero cuya ruta se pasa por parámetro.
 * Devuelve un puntero al vector y el número
 * de elementos que contiene (nr_items)
 */
int* read_vector(char* filename,
               int* nr_items) { ... }

/* even_search devuelve el número de elementos
 * pares del vector pasado como argumento
 * en en rango [start_idx:end_idx]
 */

int even_search(int* vector,
       int start_idx,int end_idx) {
  int count=0;
  int i=0;
  for (i=start_idx;i<end_idx;i++)
    if (vector[i] % 2 == 0)
      count++;
  return count;
}
```

```
int main(int argc, char* argv[]) {
    int nr_items;
    int* vector;
    pid_t pid;
    int even1=0,even2=0;
    if (argc!=2){
        fprintf(stderr,“Uso: %s <in_file>\n”,argv[0]);
        exit(1):
    }
    vector=read_vector(argv[1],&nr_items);
    if ((pid=fork())==0) {
       /*Proceso hijo */
       even1=even_search(vector,0,nr_items/2);
       exit(0);
    } else if (pid>0) {
        /* Proceso Padre */
       even2=even_search(vector,nr_items/2,nr_items);
    } else {
       fprintf(stderr,"Error en fork()\n");
       exit(2);
    }
    while(wait(NULL)!=pid) {}
    printf(“numero total de pares %d\n”,even1+even2);
    free(vector);
    return 0;
}
```

A. Explique razonadamente por qué este programa no imprime correctamente el número total de enteros pares presentes en el vector.

B. Proponga una estrategia que emplee comunicación entre procesos para conseguir que este programa multiproceso funcione correctamente (Nota: no es necesario proporcionar versiones corregidas del código. Basta con describir una aproximación a la solución.)

**Cuestión 4 (1 punto)**

El usuario root ejecuta los siguientes comandos en un SO con sistema de ficheros tipo UNIX:

```
$ mkdir /comun
$ mv /home/aurora/datos /comun/
$ ln -s /comun/datos /home/jose/db
$ ln /comun/datos /home/aurora/db
```

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

**Mapa de bits.** 1111 1100 0100 0010 ( El bit de más a la izquierda representa el primer bloque datos, bloque 1)

**Tabla de nodos-i**

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

B**loques relevantes** (los bloques que no figuran aquí contienen datos o están vacíos)

**Bloque 1**

| . | 2 |
|---|---|
| .. | 2 |
| home | 4 |
| comun | 16 |

**Bloque 2**

| . | 4 |
|---|---|
| .. | 2 |
| aurora | 10 |
| jose | 7 |

**Bloque 3**

| . | 16 |
|---|---|
| .. | 2 |
| datos | 8 |

**Bloque 4**

| . | 7 |
|---|---|
| .. | 4 |
| db | 9 |

**Bloque 15**

| . | 10 |
|---|---|
| .. | 4 |
| db | 8 |

Indique 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.

**Mapa de Bits:**_____

**Tabla de i-nodos**

| nodo-i | | | | | | | |
|---|---|---|---|---|---|---|---|
| Tipo | | | | | | | |
| Enlaces | | | | | | | |
| Directo | | | | | | | |
| Directo | | | | | | | |

**Bloques Relevantes**

**Cuestión 5. (1 punto)** Considere un sistema monoprocesador con una política de planificación de procesos de 2 niveles con realimentación. Cada nivel usa a su vez una política de planificación circular (round robin) cuyos quantos de tiempo son 2 y 4, respectivamente. Al principio hay 3 procesos en la cola del nivel 1 (máxima prioridad, quanto=2), P1, P2 y P3, en este orden. Los patrones de ejecución de los procesos son los siguientes (y se repiten indefinidamente): P1 (3-CPU,3-E/S), P2 (4-CPU,4-E/S), P3 (5-CPU, 5-E/S)

La cola del otro nivel (quanto=4) está vacía. Cuando un proceso agota su quanto, pasa a la cola de nivel inferior. Cuando acaba una operación de E/S, los procesos entran en la cola de mayor prioridad. Usando el diagrama de tiempos de la figura muestre:

a. Qué proceso está ejecutándose (utilice '**X1**' o '**X2**' para marcar el proceso en ejecución desde el nivel 1 ó 2, respectivamente, y '**O**' para marcar el proceso bloqueado por E/S) y qué procesos hay en cada nivel (utilice las filas de L1 para el primer nivel y las de L2 para el segundo), durante las 20 primeras unidades de tiempo.
b. Calcule el porcentaje de utilización de CPU y los tiempos de espera de cada proceso.

| | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1(3,3) | **X1** | | | | | | | | | | | | | | | | | | | | |
| P2(4,4) | | | | | | | | | | | | | | | | | | | | | |
| P3(5,5) | | | | | | | | | | | | | | | | | | | | | |
| L1(2) | **P2** | | | | | | | | | | | | | | | | | | | | |
| | **P3** | | | | | | | | | | | | | | | | | | | | |
| L2(4) | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |

Porcentaje de uso de CPU: _____ %
Tiempo de espera de P1: ____ unidades
Tiempo de espera de P2: ____ unidades
Tiempo de espera de P3: ____ unidades

**Cuestión 6. (1 punto)**

Considere la siguiente aplicación con POSIX Threads y sincronización **a modo de barrera** mediante semáforos.

```
#include ...
int N = 4, counter=0;
sem_t sem_barrier;
sem_t sem_mutex;
void th_function(void) {
    sem_wait(&sem_mutex);
    counter=counter+1;
    sem_post(&sem_mutex);
    /* Se hace algo */
    /* esperar al resto de threads */
    if (counter==N)  sem_post(&sem_barrier);
    sem_wait(&sem_barrier);
}
```

```
void main(void){
    pthread_t th1, th2, th3, th4;
    sem_init(&sem_barrier, 0, 0);
    sem_init(&sem_mutex, 0, 1);
    pthread_create(&th1,NULL,th_function,NULL)
    pthread_create(&th2,NULL,th_function,NULL)
    pthread_create(&th3,NULL,th_function,NULL)
    pthread_create(&th4,NULL,th_function,NULL)
    pthread_join(th1, NULL); pthread_join(th2, NULL);
    pthread_join(th3, NULL); pthread_join(th4, NULL);
    sem_destroy(&sem_barrier);
    sem_destroy(&sem_mutex);
    exit(0);
}
```

Indicar razonadamente si la implementación presenta algún problema relacionado con el progreso de los hilos. En caso afirmativo indicar una posible solución.

**SISTEMAS OPERATIVOS - PROBLEMAS**
**14 de Enero de 2019**

Nombre _____ DNI _____

Apellidos _____ Grupo _____

**Problema 1. (2 puntos)**
Se desea construir un sistema concurrente de control de acceso a un puente habilitado exclusivamente para la circulación de bicicletas, que consta de dos carriles. Tras un estudio preliminar realizado, se ha observado que el tráfico en cada sentido no es equilibrado durante todo el día, es decir, en ciertos rangos horarios el tráfico domina claramente en uno de los dos sentidos. El sistema de control de acceso a desarrollar tiene como objetivo hacer un uso eficiente de los dos carriles, activando en base al tráfico uno de los tres modos siguientes de circulación en el puente: *bidireccional* (un carril por sentido), *izquierda* o *derecha* (dos carriles activos para uno de los sentidos).

En el sistema, cada ciclista se representa como un hilo del programa concurrente, y se comporta del siguiente modo:

```
/* sentido: 0 -> izquierda, 1->derecha */
void ciclista(int sentido){
   acceder_al_puente(sentido);
   <<circular por puente>>
   salir_del_puente(sentido);
}
```

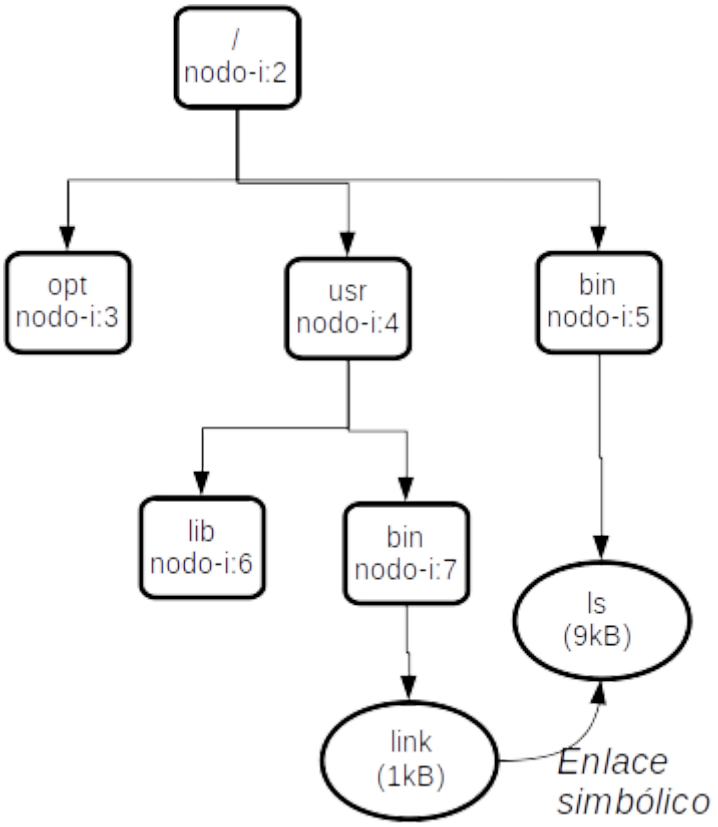
El modo de circulación del puente es *bidireccional* cuando no hay bicicletas circulando por el puente, hay bicicletas circulando en ambos sentidos o hay como mucho dos bicicletas circulando en un solo sentido. En el momento en el que se registran más de dos bicicletas circulando solo en un sentido (ninguna en el opuesto) se habilita el modo correspondiente a ese sentido -*izquierda* o *derecha*-, lo cual permitirá a los ciclistas de ese sentido usar los dos carriles simultáneamente. En este estado, cualquier ciclista que intente acceder al puente en el sentido contrario permanecerá bloqueado hasta que el estado del puente vuelva a ser *bidireccional* o del sentido correcto. Cuando el modo no sea bidireccional y haya ciclistas esperando para acceder al puente en el sentido contrario al modo activo, el sistema deberá bloquear la entrada al puente de nuevos ciclistas que vengan en el sentido activo para evitar la inanición de los que están esperando. Se permitirá la entrada a estos ciclistas cuando la entrada vuelva a ser bidireccional.

Implementar las funciones `acceder_al_puente()` y `salir_del_puente()` para imponer las restricciones de sincronización anteriormente citadas empleando mútexes, variables condicionales o semáforos, y variables compartidas. **Nota:** Además de proporcionar el código de la solución, se ha de describir claramente para qué sirve cada variable/recurso de sincronización utilizado, así como su valor inicial.

**Problema 2. (2 puntos)**

Un sistema de ficheros tipo UNIX utiliza bloques de disco de 4 Kbytes. El superbloque y el mapa de bits para los bloques del sistema de ficheros ocupa 1 bloque. La tabla de nodos-i ocupa 10 bloques y el resto de bloques se destinan a bloques de datos y de índices. Para el direccionamiento de los bloques se utilizan punteros de 32 bits y para el puntero de L/E de la la tabla intermedia de posiciones (TFA) se utiliza un entero de 24 bits. Cada nodo-i tiene 8 punteros de direccionamiento directo, 1 puntero indirecto simple y 1 puntero indirecto doble.

1. ¿Cuál es el tamaño máximo de un fichero en dicho sistema?
2. Dada la estructura de directorios y de ficheros de la siguiente figura, indicar:
    - El contenido de la tabla de nodos-i
    - El mapa de bits de bloques
    - Un contenido posible para los bloques de datos compatible con esta estructura.




*Los directorios se muestran como rectángulos con bordes redondeados y los ficheros regulares como óvalos. Link es un enlace simbólico a* `/bin/ls`

3. ¿Cuál será el espacio total ocupado en el disco?
4. Indique qué estructuras del sistema de ficheros anterior se modificarán tras la ejecución de cada uno de los siguientes comandos:

```
$ chmod +x /bin/ls
$ rm /usr/bin/link
```