Examen Final de Sistemas Operativos
Temas 1 y 2
19 de junio de 2008

NOTAS:

* La fecha de publicación de las notas, así como de revisión se notificarán por Aula Global
* Para la realización del presente examen se dispondrá de **1 hora.**
*** El examen se contesta en las hojas dadas con el enunciado.**
*** No** se pueden utilizar libros **ni** apuntes, ni usar móvil (o similar)
* Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen

---

Ejercicio 1 (3 puntos).

Responde a las siguientes cuestiones:

(a) Explica **detalladamente** el funcionamiento del siguiente código:

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

void funcion(void) {
      printf("Estoy aquí..... \n");
}

main() {
      struct sigaction act;

      act.sa_handler = funcion;
      act.sa_flags = 0;
      sigaction(SIGALRM, &act, NULL);
      act.sa_handler = SIG_IGN;
      sigaction(SIGINT, &act, NULL);

      for(;;){
            alarm(3);
            pause();
      }
}
```

(b) ¿Son siempre iguales el estado del procesador y la copia de este que reside en el BCP? Explícalo razonadamente.

(c) ¿Qué tipos de planificación existen? Explica cada uno de ellos brevemente.

(d) Enumera las distintas formas que hay de activar el sistema operativo.

Solución:

Ejercicio 2 (3,5 puntos).

Dado el siguiente conjunto de procesos planificados con round-robin con una rodaja de tiempo de 2 u.t.

Proceso	Duración	Tiempo de Llegada
A	6	0
B	4	3
C	4	5
D	6	7

La ejecución de B requiere de un operación de E/S que dura 3 u.t en el instante 3 de su ejecución.
Rellenar la tabla siguiente e indicar cuando finaliza la ejecución cada uno de ellos suponiendo que todos tienen la misma prioridad.

La ejecución será:

Ejecutado. Indicar con una lista del tipo AABBAACC… los procesos que han estado ocupando la CPU	En cola, justo antes de finalizar lo indicado en ejecutado	Tiempo Restante A	Tiempo Restante B	Tiempo Restante C	Tiempo Restante D

Proceso A finaliza en el instante ___
Proceso C finaliza en el instante ___
Proceso B finaliza en el instante ___
Proceso D finaliza en el instante ___

Solución:

La ejecución será:

Ejecutado	En cola, justo	Tiempo	Tiempo	Tiempo	Tiempo

	antes de finalizar lo indicado en ejecutado	Restante A	Restante B	Restante C	Restante D
	A	6			
AA		4			
AAAA	B	2	4		
AAAABB	AC	2	2	4	
AAAABBAA	CBD	0	2	4	6
AAAABBAACC	BD		2	2	6
AAAABBAACCB	DC		1	2	6
AAAABBAACCBDD	C		1	2	4
AAAABBAACCBDDCC	DB		1	0	4
AAAABBAACCBDDCCDD	B		1		2
AAAABBAACCBDDCCDDB	DD		0		2
AAAABBAACCBDDCCDDBDD					0

Proceso A finaliza en el instante 8
Proceso C finaliza en el instante 15
Proceso B finaliza en el instante 18
Proceso D finaliza en el instante 20

Ejercicio 3 (3.5 puntos). *El Resucitador*. Se trata de resolver un problema cuando existe una aplicación que finaliza su ejecución de forma inesperada y es necesario volver a ejecutarla de una forma automática. Para ello, se pide usando llamadas al sistema POSIX diseñar un programa que responda a los siguientes requisitos:

- Debe ejecutar el programa que se pase como argumento
- En caso de que el programa ejecutándose "muera" (finalice) por cualquier razón debe de volver a ejecutarse.
- Si se pulsa CTRL-C tanto el programa ejecutado como el propio resucitador morirán.

Solución:

```
#include <sys/types.h>
#include <stdio.h>
#include <signal.h>

#define false 0
#define true 1
int terminado = false;

void terminar(void) {
      terminado = true;
      printf("Resucitador terminado usando Ctlr-C");
}
```

```
main(int  argc, char * argv[])
{
      pid_t pid;
      int status;

      /* Programar la señal Ctrl-C */
      struct sigaction act;
      act.sa_handler = terminar;
      sigemptyset(&act.sa_mask);
      act.sa_flags = 0;
      sigaction(SIGINT, &act, NULL);



      while (!terminado){
            int mandato=argv[1];
            printf ("Ejecutando mandato %s \n",mandato);
            pid = fork();
            if (pid == 0)   { /* proceso hijo */
                  execlp(mandato,mandato,NULL);
            } else  { /* proceso padre */
                  while (pid != wait(&status));
            }
            exit(0);
      }
                                    }
```

Examen Final de Sistemas Operativos
Temas 3, 4, 5 y 6
19 de junio de 2008

NOTAS:

* La fecha de publicación de las notas, así como de revisión se notificarán por Aula Global
* Para la realización del presente examen se dispondrá de **2 horas.**
*** El examen se contesta en las hojas dadas con el enunciado.**
*** No** se pueden utilizar libros **ni** apuntes, ni usar móvil (o similar)
* Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen

---

Ejercicio 1 (2 puntos)

Responde brevemente a las siguientes cuestiones:

(a) ¿Que es un **i-nodo**? ¿Qué datos suele almacenar?

(b) Enumera las condiciones necesarias (pero no suficientes) que se deben dar para que se produzca un **interbloqueo**.

(c) Indica el valor de las variables **cont1** y **cont2** en el siguiente código. ¿Por qué toman dicho valor?

```
int contador(void) {
      static int cuenta=0;

      cuenta++;
      return cuenta;
}

main() {

int cont1,cont2;

cont1=contador();
cont2=contador();

}
```

(d) ¿Cuál es la razón de que una llamada al servicio **wait** (cuando se usan mutex y variables condicionales) se haga dentro de un bucle while?

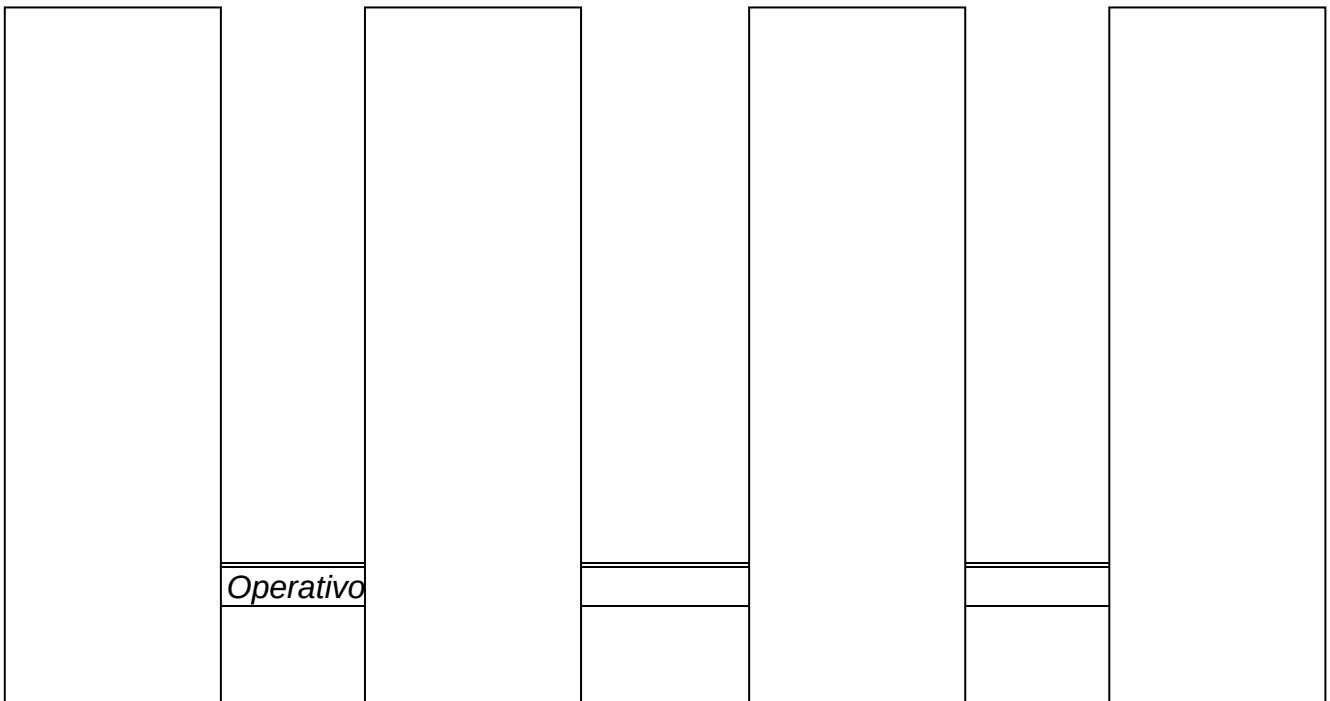
Solución

Ejercicio 2 (2.5 puntos)

Atendiendo al siguiente código:

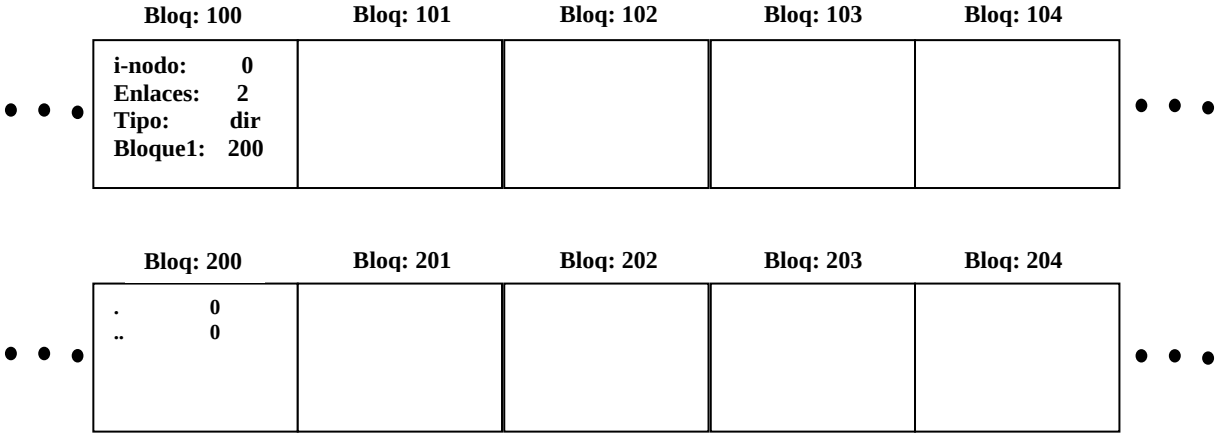
```
unsigned int a=1;
float b=2,c;

void f1(int d) {
      float i;
      int j;
      static g;

      b = i + j + 5;
      .......
B     f2(j)
      return;
}

void f2(int w) {
      int i;
      static f = 2;

      a=a+i+w+f;
      .......
      return;
}

main (int argc, char **argv) {
      int *p;
      char *q, *w;

A     p = (int *) malloc (512)
      q = (char *) malloc (1024)

      f1(b);

C     w=mmap (fichero)
      .......
      free (p)
      .......
      free (q)
      free (w)
      ....
      exit (0)
}
```

a. Indica como evolucionan las regiones o segmentos de memoria del programa cuando se lanza su ejecución, y en los puntos del código indicados con flechas rellenando las siguientes figuras. Indica así mismo el contenido de cada segmento.



Ejercicio 3 (2,5 puntos)

Se dispone de un Sistema de Ficheros estilo UNIX. El tamaño de bloque es de 512 bytes. El i-nodo 0 corresponde al directorio raíz del sistema.

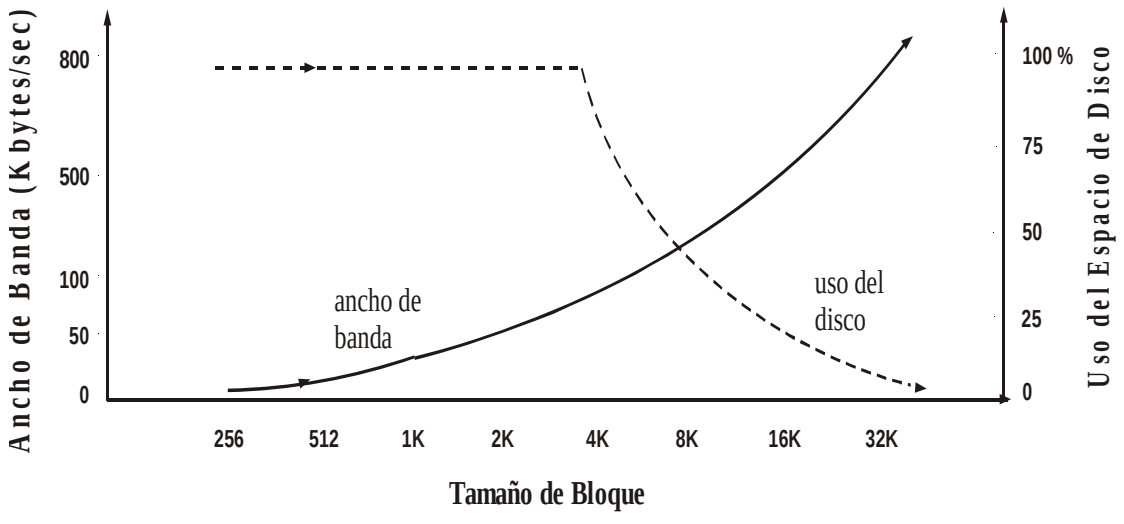


Se pide:

1.- **Explicar detalladamente y representar** el estado en el que queda el sistema de ficheros después de ejecutar los siguientes mandatos. Tener en cuenta que el mandato anterior puede afectar la ejecución del siguiente: **(2 puntos)**

a) *mkdir /home.*
b) *touch /home/prueba.txt (crea un fichero si no existe y modifica el atributo de fecha de último acceso si existe)*
c) *ln –s /home/prueba.txt /prueba2.txt*
d) *rm –r /home (borrado recursivo del directorio home)*

2.- **Dado el siguiente gráfico** donde se muestra la relación entre el tamaño de bloque la tasa de transferencia y del % de utilización de disco. Justificar entre que rangos debe moverse el tamaño de bloque para obtener una buena tasa de transferencia sin perjudicar el grado de utilización de disco. **(0,5 Puntos)**



Solución:

A) mkdir /home.

a) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200)
b) Comprobar que no existe entrada con nombre home dentro del bloque de datos. Si es así se asigna un nuevo inodo (inodo 1- bloque 101) y se define como tipo directorio
c) Se asigna un bloque de datos nuevo (bloque de datos 201) y se crean dos entradas nuevas apuntando a el mismo (inodo 1) y al inodo padre (inodo 1)
d) Se actualiza el número de enlaces del inodo padre con un enlace más

e)

	Bloq: 100	Bloq: 101	Bloq: 102	Bloq: 103	Bloq: 104
	i-nodo: 0 Enlaces: 3 Tipo: dir Bloque1: 200	i-nodo: 1 Enlaces: 2 Tipo: dir Bloque1: 201			

	Bloq: 200	Bloq: 201	Bloq: 202	Bloq: 203	Bloq: 204
	. 0 .. 0 home 1	. 0 .. 0			

B) touch /home/prueba.txt (crea un fichero si no existe y modifica el atributo de fecha de último acceso si existe)

a) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200), como ahí se encuentra la entrada de 'home' (inodo 1) no hay que leer más bloques.
b) Leer el inodo de home (inodo 1 - bloque 101), se ve que es un directorio y se lee el primer bloque de datos (bloque 204). Se añade la entrada prueba.txt apuntando a un nuevo inodo (inodo 3- bloque 103)
c) Se establece el inodo 3 como de texto y se le asigna un bloque de datos nuevo y vacio

Bloq: 100	Bloq: 101	Bloq: 102	Bloq: 103	Bloq: 104
i-nodo: 0 **Enlaces: 3** **Tipo: dir** **Bloque1: 200**	**i-nodo: 1** **Enlaces: 2** **Tipo: dir** **Bloque1: 201**	**i-nodo: 2** **Enlaces: 1** **Tipo: file** **Bloque1: 202**		

Bloq: 200	Bloq: 201	Bloq: 202	Bloq: 203	Bloq: 204
. 0 **.. 0** **home 1**	**. 0** **.. 0** **Prueba.txt 2**			

C) ln –s /home/prueba.txt /prueba2.txt

a) **Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200), como ahí se encuentra la entrada de 'home' (inodo 1) no hay que leer más bloques.**
b) **Leer el inodo de home (inodo 1 - bloque 101), se ve que es un directorio y se lee el primer bloque de datos (bloque 201). Se lee la entrada 'prueba.txt' (inodo 2) no hay que leer más bloques.**
c) **Leer el inodo de 'prueba.txt' (inodo 1 - bloque 101), se ve que es un fichero y se comprueba que existe.**
d) **Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200). Ahí se comprueba que no existe y se crea la entrada prueba2.txt (inodo 3 – bloque 103) estableciendo el inodo como tipo enlace simbólico.**
e) **Se asigna un nuevo bloque de datos y dentro de él se establece la ruta a /home/prueba.txt**

Bloq: 100	Bloq: 101	Bloq: 102	Bloq: 103	Bloq: 104
i-nodo: 0 **Enlaces: 3** **Tipo: dir** **Bloque1: 200**	**i-nodo: 1** **Enlaces: 2** **Tipo: dir** **Bloque1: 201**	**i-nodo: 2** **Enlaces: 1** **Tipo: file** **Bloque1: 202**	**i-nodo: 3** **Enlaces: 1** **Tipo: Symbol** **Bloque1: 203**	

Bloq: 200	Bloq: 201	Bloq: 202	Bloq: 203	Bloq: 204
. 0 **.. 0** **home 1** **prueba2.txt 3**	**. 0** **.. 0** **Prueba.txt 2**		**/home/prueba.t xt**	

D) rm –r /home (borrado recursivo del directorio home)

a) Leer el inodo del directorio raiz (inodo 0 - bloque 100) y luego se lee el primer bloque de datos de dicho directorio (bloque 200), como ahí se encuentra la entrada de 'home' (inodo 1) no hay que leer más bloques.
b) Leer el inodo de home (inodo 1 - bloque 101), Si

a. **La entrada es un directorio y se lee el primer bloque de datos (bloque 2041. Por cada entrada se ejecutan los pasos a y b aplicados a cada directorio en vez del directorio raiz.**
b. **La entrada es un fichero se elimina el fichero accediendo a sus bloques de datos y marcándolos como libres en el volumen (mapa de bits) y eliminando el inodo de la lista enlazada de inodos libres (y en el mapa de bits de inodos si lo hubiera). En este caso se elimina el fichero prueba.txt que se encuentra dentro del directorio home.**

c) **Se elimina el directorio home puesto (al estar ya vacio por la iteración de pasos a y b). Para ello se elimina el i-nodo 1 y el bloque de datos 101.**
d) **Se elimina la entrada home dentro del bloque de datos del directorio raíz (bloque 100)**
e) **Se decrementa en 1 el número de enlaces al directorio raíz**
f) **El enlace simbólico sigue permaneciendo puesto que es un fichero y no estaba dentro del directorio home**

Bloq: 100	Bloq: 101	Bloq: 102	Bloq: 103	Bloq: 104
i-nodo: 0 Enlaces: 2 Tipo: dir Bloque1: 200			i-nodo: 3 Enlaces: 1 Tipo: Symbol Bloque1: 203	

Bloq: 200	Bloq: 201	Bloq: 202	Bloq: 203	Bloq: 204
. 0 .. 0 home prueba2.txt 3			/home/prueba.txt	

2.- Lo que se pide en este ejercicio es razonar la respuesta que se de puesto que no existe una respuesta que sea totalmente correcta/incorrecta:

Se observa que el rendimiento optimo a nivel de ancho de banda se encuentra con bloques de datos grandes. Cuanto mayor sea el bloque de datos más nos aproximamos a la tasa de transferencia óptima definida por el disco de almacenamiento. También se observa que aumenta la fragmentacíon interna provocada por la diferencia entre el tamño real de un fichero y el bloque de datos asignado.
Elegir un valor óptimo depende en cierta medida de los tipos de datos que vayamos almacenar (principalmente el tamño de los ficheros que se usen). En general hay que buscar un equilibrio tasa de transferencia y fragmentación interna.
En el esquema presentado se observa que el tamaño de bloque óptimo se encuentra en un rango que varia entre 4kb y 32 kb. 16 kb se puede observa como un buen compromiso entre ambos parámetros.

Ejercicio 4 (3 puntos)

Realizar un programa que simule el comportamiento de un sistema de 2 ascensores. Existirá 1 **thread** por cada ascensor y el **thread** padre que actuará como controlador.
Habrá, al menos, 2 variables compartidas por cada ascensor, en una se indicará la planta en la que está el ascensor y en la otra la planta a la que debe trasladarse.
Los ascensores tendrán que estar esperando a que la planta a la que desean trasladarse sea diferente de aquella en la que están. En ese momento incrementarán o decrementarán en una unidad, cada segundo, el valor de la planta en la que están hasta que lleguen a la planta destino. Cuando llegue se lo comunicará al controlador de la forma que el alumno considere adecuada.
El controlador pedirá al usuario una planta de destino y después de comprobar en que planta se encuentra cada uno de los ascensores parados colocará en la variable destino del elegido la nueva planta de destino. Si un ascensor se encuentra en movimiento no será elegible para mandarlo a la nueva planta.
El acceso a las variables compartidas se realizará siempre utilizando **mutex**, y cuando sea adecuado también se utilizarán variables condicionales.
Los **mutex** deben bloquear el acceso el menor tiempo posible.

Se puede modificar el código siguiente, añadiendo en los fragmentos numerados o hacer un código completamente nuevo.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int estancia[]={0,0}; //plantas en las que se encuentra cada ascensor
int destino[]={0,0}; //plantas de destino de cada ascensor
int moviendose[]={0,0}; // indica si el ascensor se mueve (1) o está
parado (0)
                        // se pone a 1 cuando empieza a moverse y a 0
cuando llega a su destino

pthread_mutex_t mutexe[2]; // para el acceso a las plantas de posición
pthread_mutex_t mutexd[2]; // para el acceso a las plantas de destino

......-1-......


void moveraotraplanta(){
  sleep(1);
}

void *ascensor(int *numasc) { // El parámetro indica el número de
ascensor
  int i ,pos = 0,inc=0,asc;

  asc=*numasc;
  printf ("Arrancado ascensor %d\n", asc);

   while (1){
```

```
      ......-2-......    // Mira en que planta está


      pos=estancia[asc];

      ......-3-......



    while (estancia[asc] != destino[asc]){

      ......-4-......


    }

      ......-5-......


  }
    pthread_exit(0);
}

void *controlador(void *kk) {   //código del controlador
  int destinoUsu;
  int distancia0, distancia1; //distancia a la planta destino a la que
se encuentra cada ascensor
  int usar0=0, usar1=0; //indican que ascensor se va a utilizar para
que se dirija a la planta elegida por el usuario. Estarán a 0 si no se
puede usar el ascensor correspondiente. Si un ascensor está en
movimiento no se puede usar, si ambos están parado se elige el más
cercano.

  while (1){
    printf ("Introducir planta:");
    scanf ("%d", &destinoUsu);
    do {
      usar0=0;
      usar1=0;
      pthread_mutex_lock(&mutexe[0]);
        if ( ......-6-...... ) {
          distancia0=abs(estancia[0]-destinoUsu);
          usar0=1;
        }
      pthread_mutex_unlock(&mutexe[0]);
      pthread_mutex_lock(&mutexe[1]);
        if ( ......-7-...... ){
          distancia1=abs(estancia[1]-destinoUsu);
          usar1=1;
        }
```

```
      pthread_mutex_unlock(&mutexe[1]);

      } while (!usar0 && !usar1); // ninguno se puede utilizar, los 2
      están moviéndose

    if (usar1 && usar0) // los 2 están parados
       if (distancia0 > distancia1 )
         usar0=0;
       else
         usar1=0;

    if (usar1){ // envío al 1
      pthread_mutex_lock(&mutexd[1]);
        destino[1]=destinoUsu;

        ......-8-......


        pthread_mutex_unlock(&mutexd[1]);
    }
    else {  // envío al 0
      pthread_mutex_lock(&mutexd[0]);
        destino[0]=destinoUsu;

        ......-9-......


      pthread_mutex_unlock(&mutexd[0]);
    }
  }
  pthread_exit(0);
}


main(int argc, char *argv[]){
    pthread_t th1, th2;
    int asc1=0,asc2=1;

    pthread_mutex_init(&mutexe[0], NULL);
    pthread_mutex_init(&mutexe[1], NULL);
    pthread_mutex_init(&mutexd[0], NULL);
    pthread_mutex_init(&mutexd[1], NULL);
    ......-10-......



    pthread_mutex_destroy(&mutexe[0]);
    pthread_mutex_destroy(&mutexe[1]);
    pthread_mutex_destroy(&mutexd[0]);
    pthread_mutex_destroy(&mutexd[1]);
    ......-11-......
```

```
    exit(0);
}
```

Solución

Valoración de cada subapartado

- Subapdo 1: 0,25
- Subapdo 2: 0,25
- Subapdo 3: 0,5
- Subapdo 4: 0,5
- Subapdo 5: 0,5
- Subapdo 6 + Subapdo 7: 0,25
- Subapdo 8 + Subapdo 9: 0,25
- Subapdo 6 + Subapdo 7: 0,5

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int estancia[]={0,0}; //plantas en las que se encuentra cada ascensor
int destino[]={0,0}; //plantas de destino de cada ascensor
int moviendose[]={0,0}; //indica si el ascensor se mueve (1) o está
parado (0)
  // se pone a 1 cuando empieza a moverse y a 0 cuando llega a su
destino

pthread_mutex_t mutexe[2]; // para el acceso a las plantas de posición
pthread_mutex_t mutexd[2]; // para el acceso a las plantas de destino
pthread_cond_t parado[2]; //para señalizar que el ascensor debe
moverse

void moveraotraplanta(){
  sleep(1);
}
void *ascensor(int *numasc)  {   /* código del ascensor, el parámetro
indica el número de ascensor*/
  int i ,pos = 0,inc=0,asc;
  asc=*numasc;
printf ("Arrancado ascensor %d\n", asc);
  while (1){
    pthread_mutex_lock(&mutexe[asc]);        /* Mira en que planta
está */
      pos=estancia[asc];  //libero pues sólo este thread modifica la
variable
    pthread_mutex_unlock(&mutexe[asc]);
    pthread_mutex_lock(&mutexd[asc]);
printf("ascensor %d, espero destino\n",asc);
    while (pos == destino[asc])
```

```
      pthread_cond_wait(&parado[asc], &mutexd[asc]);
printf("ascensor %d, nuevo destino:%d\n",asc, destino[asc]);
    if (pos > destino[asc])
      inc=-1;
    else
      inc=1;
    pthread_mutex_lock(&mutexe[asc]);
    moviendose[asc]=1;
    while (estancia[asc] != destino[asc]){
      estancia[asc]=estancia[asc]+inc;
printf ("ascensor %d está en planta %d\n",asc, estancia[asc]);
      pthread_mutex_unlock(&mutexe[asc]);
      moveraotraplanta();
      pthread_mutex_lock(&mutexe[asc]);
    }
    moviendose[asc]=0;
    pthread_mutex_unlock(&mutexe[asc]);
    pthread_mutex_unlock(&mutexd[asc]);
  }
    pthread_exit(0);
}

void *controlador(void *kk) {   /* código del controlador */
  int destinoUsu;
  int distancia0, distancia1; //distancia a la planta destino a la que
se encuentra cada ascensor
  int usar0=0, usar1=0; //indican que ascensor se va a utilizar para
que se dirija a la planta elegida por el usuario. Estarán a 0 si no se
puede usar el ascensor correspondiente. Si un ascensor está en
movimiento no se puede usar, si ambos están parado se elige el más
cercano.

  while (1){
    printf ("Introducir planta:");
    scanf ("%d", &destinoUsu);
    do {
      usar0=0;
      usar1=0;
      pthread_mutex_lock(&mutexe[0]);
        if (!moviendose[0]){
          distancia0=abs(estancia[0]-destinoUsu);
          usar0=1;
        }
      pthread_mutex_unlock(&mutexe[0]);
      pthread_mutex_lock(&mutexe[1]);
        if (!moviendose[1]){
          distancia1=abs(estancia[1]-destinoUsu);
          usar1=1;
        }
      pthread_mutex_unlock(&mutexe[1]);

    } while (!usar0 && !usar1); // ninguno se puede utilizar, los 2
están moviéndose

    if (usar1 && usar0) // los 2 están parados
```

```
        if (distancia0 > distancia1)
          usar0=0;
        else
          usar1=0;

    if (usar1){ // envio al 1
printf ("Controlador:Mando al ascensor 1 a la planta %d\n",
destinoUsu);
      pthread_mutex_lock(&mutexd[1]);
        destino[1]=destinoUsu;
        pthread_cond_signal(&parado[1]);
      pthread_mutex_unlock(&mutexd[1]);
    }
    else {  // envío al 0
printf ("Controlador:Mando al ascensor 0 a la planta %d\n",
destinoUsu);
      pthread_mutex_lock(&mutexd[0]);
        destino[0]=destinoUsu;
        pthread_cond_signal(&parado[0]);
      pthread_mutex_unlock(&mutexd[0]);
    }
  }
  pthread_exit(0);
}


main(int argc, char *argv[]){
    pthread_t th1, th2;
    int asc1=0,asc2=1;

    pthread_mutex_init(&mutexe[0], NULL);
    pthread_mutex_init(&mutexe[1], NULL);
    pthread_mutex_init(&mutexd[0], NULL);
    pthread_mutex_init(&mutexd[1], NULL);
    pthread_cond_init(&parado[0], NULL);
    pthread_cond_init(&parado[1], NULL);
    pthread_create(&th1, NULL, ascensor, (void *) &asc1);
    pthread_create(&th1, NULL, ascensor, (void *) &asc2);
    pthread_create(&th2, NULL, controlador, NULL);

    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    pthread_mutex_destroy(&mutexe[0]);
    pthread_mutex_destroy(&mutexe[1]);
    pthread_mutex_destroy(&mutexd[0]);
    pthread_mutex_destroy(&mutexd[1]);
    pthread_cond_destroy(&parado[0]);
    pthread_cond_destroy(&parado[1]);

    exit(0);
}
```