Examen Extraordinario de Sistemas Operativos - 18 de Junio de 2013

Grado en Ingeniería Informática

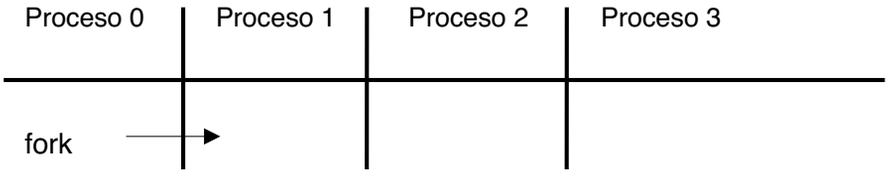
EXAMEN RESUELTO

* No se pueden utilizar libros, apuntes ni aparatos digitales.
* Será necesario presentar el DNI o carnet universitario para realizar la entrega del examen

Problema 2

Rellenar en una tabla similar a la que se muestra, las llamadas al sistema operativo (fork, close, .) que se producirán para ejecutar el siguiente mandato:

ls lalcat<f1.txt> f2.txt



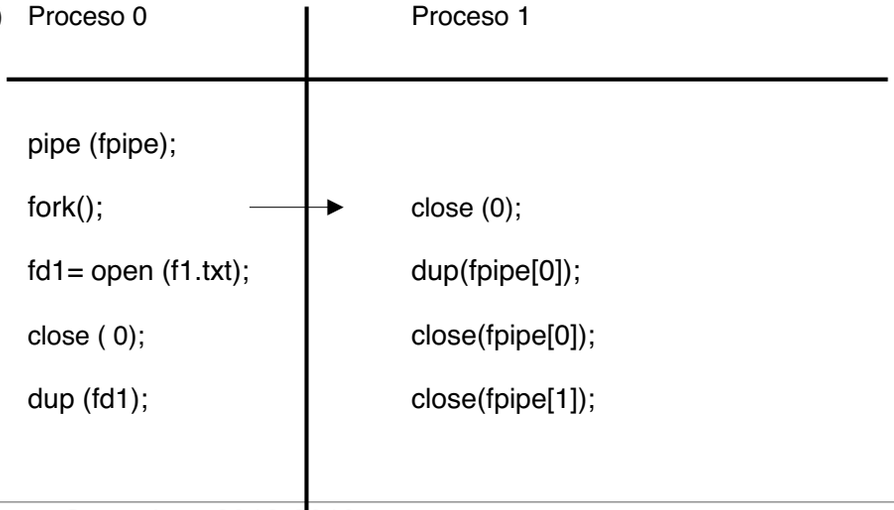
a) Dado el siguiente código:

```
for (i=1; i<4; i++){
  pid=fork();
     if (pid==0){ //Hijo
               printf ("Hijo %d, pid =%d, ppid=%d\n",i, getpid(),getppid());
               exit(0);
     }
  else
     printf ("El padre ha creado %d hijos \n",i);
     exit (0);
}
```

Suponiendo que el PID de este proceso es 700, indicar la jerarquía de procesos generada identificando cada proceso con un valor PID consecutivo al del padre, y escribir los mensajes que aparecerán en pantalla.

SOLUCIÓN:

a)



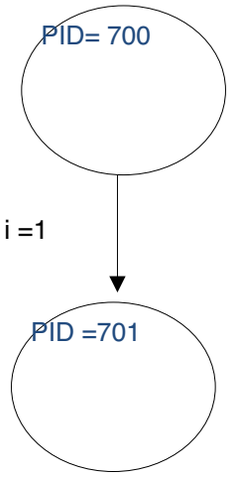
Grado en Ingeniería Informática

EXAMEN RESUELTO

```
close(1);                    fd2=open(f2.txt);
dup(fpipe[1]);               close(1);
close(fpipe[0];              dup(fd2);
close(fpipe[1]);             exec(cat,.);
exec(ls,-la,.);              exit(1);
exit(1);
```

Esta es una de las posibles soluciones existen más

a)



Hijo 1, pid=701, ppid=700

El padre ha creado 1 hijos

Esta es una de las posibles soluciones, el orden de los mensajes puede variar

Problema 3 Se desea realizar un programa que, utilizando threads, mutex y variables condicionales (no se podrán utilizar sleeps ni similares para conseguir sincronización), simule el funcionamiento de un ascensor con las siguientes características:

- En el ascensor pueden ir entrando personas siempre que no se sobrepase el PESOMAXIMO (200kg).
- Cada persona es un thread creado con un peso aleatorio.
- El ascensor se pone en marcha solo si el peso en su interior está entre 150 y 200 kg.
- Cada thread de persona que suba al ascensor escribirá un mensaje con el texto Sube persona con peso " seguida de su peso.
- Y cuando el ascensor alcance la capacidad adecuada escribirá un mensaje con el texto Viajando persona con peso" seguida de su peso.
- Cuando finalice cada viaje, la última persona que salga del ascensor escribirá Fin del viaje

Ejemplo de ejecución:

Grado en Ingeniería Informática

EXAMEN RESUELTO

```
Sube persona con peso 16
Sube persona con peso 45
Sube persona con peso 43
Sube persona con peso 25
Sube persona con peso 36
Viajando  persona con peso 36
Viajando  persona con peso 16
Viajando  persona con peso 45
Viajando  persona con peso 43
Viajando  persona con peso 25
Fin del viaje
Sube persona con peso 62
Sube persona con peso 18
Sube persona con peso 37
Sube persona con peso 60
Viajando  persona con peso 60
Viajando  persona con peso 62
Viajando  persona con peso 18
Viajando  persona con peso 37
Fin del viaje
```

Para facilitar el ejercicio se proporciona el siguiente esqueleto de ejecución que debe completar el alumno en las zonas indicadas como COMPLETAR N.

Responder indicando únicamente el código a añadir en esas secciones COMPLETAR N.

Esqueleto

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define TRUE        1
#define FALSE       0
#define NUMPERSONAS  50
#define PESOMAXIMO  200

pthread_mutex_t mtx;     /* mutex para controlar el acceso al ascensor*/
pthread_cond_t espera;   /* controla la espera de las persona*/

pthread_mutex_t mtxMoverse; /*mutex para controlar cuando se mueve el ascensor*/
pthread_cond_t esperaMoverse; /*controla cuando se mueve el ascensor*/

int pesoAscensor=0;      //Peso de la gente que está dentro del ascensor
int lleno=FALSE;      //Peso de la gente que está dentro del ascensor
int numViaje=0;      //Número del viaje
int contPersonasSubidas=0; /*Indica cuantas personas están subidas al ascensor*/

pthread_mutex_t mtxPeso;     /* mutex para controlar el acceso al peso*/
pthread_cond_t esperaPeso;   /* controla la espera en el índice */
```

Grado en Ingeniería Informática

EXAMEN RESUELTO

```
int hiloespera=1;

void  *persona(void *peso)  {
  int mipeso;

//Completar guardando en la variable local mipeso el parámetro recibido.
                    COMPLETAR 1

pthread_mutex_lock(&mtx);
   //Completar haciendo esperar si quieren entrar y el ascensor está lleno o la persona que
desea entrar tiene un peso excesivo
                    COMPLETAR 2
      contPersonasSubidas++;
      printf ("Sube persona con peso %d\n", mipeso);
//Completar haciendo esperar a las personas que ya han subido hasta que se alcance el peso
mínimo (150Kg)
                    COMPLETAR 3
      lleno=TRUE;
      printf ("Viajando  persona con peso %d\n", mipeso);
      pthread_cond_signal(&esperaMoverse);   //Despertamos a los que están esperando
dentro del ascensor
      contPersonasSubidas - -;  //Sale la persona
//Completar para que el último en subir indique que el ascensor está libre
                    COMPLETAR 4
   pthread_mutex_unlock(&mtx);
   pthread_exit(0);
}


main(int argc, char *argv[]){
   pthread_t th[NUMPERSONAS];
   int i, peso;

   pthread_mutex_init(&mtx, NULL);
   pthread_cond_init(&espera, NULL);
   pthread_mutex_init(&mtxPeso, NULL);
   pthread_cond_init(&esperaPeso, NULL);
   pthread_cond_init(&esperaMoverse, NULL);

  for (i=0; i<NUMPERSONAS; i++){
      pthread_mutex_lock(&mtxPeso);        /* acceder al peso */
      peso=10+random()%80;  //Peso entre 10 y 90 kg cada persona
      pthread_create(&th[i], NULL, persona, &peso);
//Completar haciendo esperar al main hasta que el thread haya tomado su peso y creando el
thread con la instrucción: pthread_create(&th[i], NULL, persona, &peso);
                    COMPLETAR 5
  }
  for (i=0; i<NUMPERSONAS; i++)
   pthread_join(th[i], NULL);
```

```
    pthread_mutex_destroy(&mtx);
    pthread_cond_destroy(&espera);
    pthread_mutex_destroy(&mtxPeso);
    pthread_cond_destroy(&esperaPeso);
    pthread_cond_destroy(&esperaMoverse);

    exit(0);
}
```

SOLUCIÓN

En negrita lo que tienen que rellenar en las zonas de completar

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX        1       // número máximo de veces que viaja cada persona
#define TRUE       1
#define FALSE      0

#define NUMPERSONAS  50
#define PESOMAXIMO  200

pthread_mutex_t mtx;     /* mutex para controlar el acceso al ascensor*/
pthread_cond_t espera;   /* controla la espera de los persona*/

pthread_mutex_t mtxMoverse; /*mutex para controlar cuando se mueve el ascensor*/
pthread_cond_t esperaMoverse; /*controla cuando se mueve el ascensor*/

int pesoAscensor=0;      //Peso de la gente que está dentro del ascensor
int lleno=FALSE;      //Peso de la gente que está dentro del ascensor
int numViaje=0;      //Número del viaje
int contPersonasSubidas=0; /*Indica cuantas personas están subidas al ascensor*/

pthread_mutex_t mtxPeso;     /* mutex para controlar el acceso al peso*/
pthread_cond_t esperaPeso;   /* controla la espera en el índice */
int hiloespera=1;

void  *persona(void *peso)  {
  int mipeso,j;

  pthread_mutex_lock(&mtxPeso);        /* acceder al peso */
     hiloespera=0;
     mipeso=*((int *) peso);
     pthread_cond_signal(&esperaPeso);
   pthread_mutex_unlock(&mtxPeso);        /* fin acceder al peso */

   pthread_mutex_lock(&mtx);
```

```
      while (pesoAscensor + mipeso > PESOMAXIMO || lleno )
        pthread_cond_wait(&espera, &mtx);
      pesoAscensor = pesoAscensor +  mipeso; //Subo al ascensor
      contPersonasSubidas++;
      printf ("Sube persona con peso %d\n", mipeso);

      while (pesoAscensor < 3*PESOMAXIMO/4 )
        pthread_cond_wait(&esperaMoverse, &mtx);
      lleno=TRUE;
      printf ("Viajando  persona con peso %d\n", mipeso);
//sleep(1);
      pthread_cond_signal(&esperaMoverse);

      contPersonasSubidas--;  //Sale la persona
      if (contPersonasSubidas ==0) { //El último indica que pueden pasar los demás.
       lleno=FALSE;
       pesoAscensor=0;
        printf ("Fin viaje %d\n", numViaje++);
        pthread_cond_broadcast(&espera);
      }
   pthread_mutex_unlock(&mtx);

   pthread_exit(0);
}


main(int argc, char *argv[]){
   pthread_t th[NUMPERSONAS];
   int i, peso;

   pthread_mutex_init(&mtx, NULL);
   pthread_cond_init(&espera, NULL);
   pthread_mutex_init(&mtxPeso, NULL);
   pthread_cond_init(&esperaPeso, NULL);
   pthread_cond_init(&esperaMoverse, NULL);

  for (i=0; i<NUMPERSONAS; i++){
   peso=10+random()%80;  //Peso entre 10 y 90 kg cada persona
   pthread_mutex_lock(&mtxPeso);        /* acceder al peso */
   pthread_create(&th[i], NULL, persona, &peso);
     while (hiloespera==1)
          pthread_cond_wait(&esperaPeso, &mtxPeso ); /* se espera */
     hiloespera=1;
   pthread_mutex_unlock(&mtxPeso);        /* acceder al peso */
  }
  for (i=0; i<NUMPERSONAS; i++)
   pthread_join(th[i], NULL);

   pthread_mutex_destroy(&mtx);
```

```
    pthread_cond_destroy(&espera);
    pthread_mutex_destroy(&mtxPeso);
    pthread_cond_destroy(&esperaPeso);
    pthread_cond_destroy(&esperaMoverse);

    exit(0);
}
```

Problema 4 Dado el siguiente código:

```
int total;
int inicial=0;

int main(int argc, char **argv)
{
        int *punt;
        int i;
                /*A*/
        punt = malloc (10*sizeof(int));

        for (i=0; i<=9; i++){
                punt[i]=1;
                inicial++;
        }
        total=inicial;
        printf("Total...%d Inicial...%d\n",total,inicial);
            /*B*/
        return(1);
}
```

a. Indique el estado de la memoria detallado (regiones y contenido) en el instante que representa el comentario /*A*/

b. Indique el estado de la memoria detallado (regiones y contenido) en el instante que representa el comentario /*B*/

c. Indique y defina al menos dos regiones de memoria que no estén representadas en este problema.

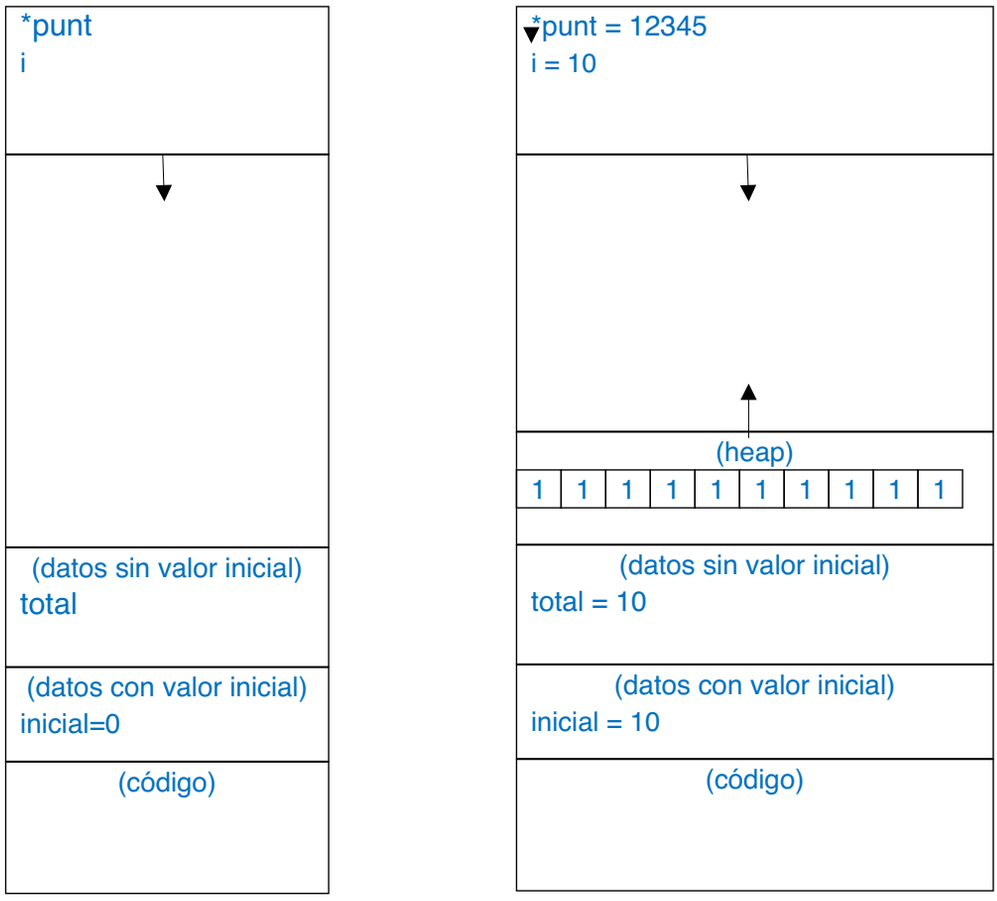
SOLUCIÓN:

a. Indique el estado de la memoria en el instante que representa el /*A*/

(pila)
variables entorno
argv
argc

b. Indique el estado de la memoria en el instante que representa el /*B*/

(pila)
variables entorno
argv
argc



c. Indique y defina al menos dos regiones de memoria que no estén representadas en este problema.

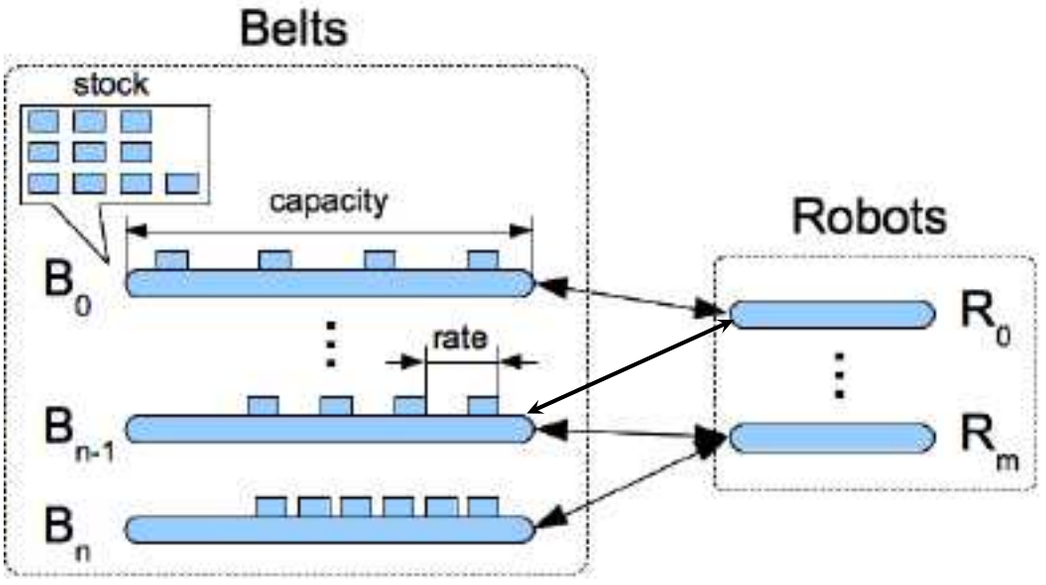
Otras regiones de memoria que podría tener un proceso y que no se representan en este problema son:

- Archivo proyectado en memoria: región de memoria asociada a un archivo proyectado desde almacenamiento secundario.
- Memoria compartida: para poder contar con un espacio de almacenamiento común entre dos o más procesos.
- Pilas de threads: pilas específicas de cada thread que pueda lanzar un proceso.
- Cargas de bibliotecas dinámicas: región asociada al código y datos de una biblioteca dinámica usada por el proceso.

Problema 5

Se desea programar una fábrica que está compuesta por 3 cintas transportadoras y 2 robots. Cada cinta tiene un productor que pone piezas cada 5 segundos. Los robots realizan el montaje

final cogiendo elementos de una o más cintas. Vea el ejemplo de la figura.



Una cinta tiene una capacidad máxima, por lo que el productor debe parar si la cinta se llena.

La cinta es un elemento compartido entre productor y consumidor.

El productor se debe simular con un hilo que produce para cada cinta y debe dormir 5 segundos hasta volver a insertar un nuevo elemento en la misma. Produce siempre que puede y no tiene límite de número de productos.

Los robots de ensamblado producen elementos siempre que pueden y no tienen límite de número. Un robot de ensamblado se caracteriza por las cintas de las que tiene que coger elementos para producir un item. Cuando tiene las piezas tarda en producir cada elemento 8 segundos.

En este caso se quiere crear una configuración fija con 3 cintas y 2 robots. De forma que Robot0 coge de cinta0 y cinta1 y Robot1 coge de cintas 1 y 2 para poder producir.

Se pide:

a) Programar sistema de productores, robots y de cintas de forma que todo debe estar sincronizado para iniciar la producción en el mismo instante.

b) Programar el sistema productor-consumidor sobre las cintas.

c) Programar el sistema de sincronización cintas-robots de forma que produzcan adecuadamente. Cada robot debe esperar por las cintas que necesite antes de producir y no debe haber bloqueos.

SOLUCION

Grado en Ingeniería Informática

EXAMEN RESUELTO

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

// Constantes que almacenan las caracteristicas de los elementos de la
fabrica
const int DORMIR_PRODUCTOR = 5;
const int DORMIR_ROBOT = 8;
const int CAPACIDAD_CINTA = 10;
const int NUM_CINTAS = 3;
const int NUM_ROBOTS = 2;

// Estructura para guardar los datos especificos de cada cinta
struct datos_cinta{
        int id;
        int robot1;
        int robot2;
};

// Estructura para guardar los datos especificos de cada robot
struct datos_robot{
        int id;
        int cinta1;
        int cinta2;
};

pthread_t *threads; // Array que guarda los identificadores de todos
los hilos

pthread_mutex_t mutex; // Mutex que protege el numero de elementos en
las cintas
pthread_barrier_t barrera; // Barrera para comenzar la produccion de
forma simultanea

int *elementos_por_cinta; // Array de los elementos que hay
actualmente sobre cada una de la cintas



pthread_cond_t *cinta_llena;  // Array de variables condicionales que
controlan cuando estan las cintas llenas
pthread_cond_t *cinta_vacia;  // Array de variables condicionales que
controlan cuando estan las cintas vacias

/*
 * Funcion que simula el comportamiento de una cinta
 */
void * hilo_cinta(void * datos){

        struct datos_cinta *cinta;
        cinta=(struct datos_cinta *) datos;



        pthread_barrier_wait(&barrera);
```

```
        // La cinta comienza la produccion a la vez que el resto

        int i;
        while(1){
                // Se protege el acceso a elementos_por_cinta
                pthread_mutex_lock(&mutex);
                // Mientras la cinta este llena, espera para producir
                while (elementos_por_cinta[cinta->id] == CAPACIDAD_
CINTA){
                        printf("La cinta %d esta llena \n",cinta->id);
                        pthread_cond_wait(&cinta_llena[cinta->id], &
mutex);
                }

                // Produce un nuevo elemento
                elementos_por_cinta[cinta->id] ++;

                // Se avisa a los robots que esperan por la cinta de
que no esta vacia
                pthread_cond_broadcast(&cinta_vacia[cinta->id]);
                printf("La cinta %d ha producido y duerme. Hay %d
elementos en la cinta. \n",cinta->id,elementos_por_cinta[cinta->id]);
                pthread_mutex_unlock(&mutex);
                sleep(DORMIR_PRODUCTOR);
        }

        free(cinta);

        pthread_exit(0);
}

/*
 * Funcion que simula el comportamiento de un robot
 */
void * hilo_robot(void * datos){

        struct datos_robot *robot;
        robot=(struct datos_robot *) datos;

        // Robot creado. Esperar al resto
        pthread_barrier_wait(&barrera);
        // El robot comienza la produccion a la vez que el resto


        while(1){
                // Se protege el acceso a elementos_por_cinta
                pthread_mutex_lock(&mutex);

                // Si alguna de las cintas esta vacia, se espera por
ella
```

```
                while (elementos_por_cinta[robot->cinta1] == 0 ||
elementos_por_cinta[robot->cinta2] == 0){
                        if(elementos_por_cinta[robot->cinta1] == 0){
                                printf("El robot %d espera por cinta %d
\n",robot->id,robot->cinta1);
                                pthread_cond_wait(&cinta_vacia[robot->
cinta1], &mutex);
                        }else{
                                printf("El robot %d espera por cinta %d
\n",robot->id,robot->cinta2);
                                pthread_cond_wait(&cinta_vacia[robot->
cinta2], &mutex); /* se bloquea */
                        }
                }

                // Se quita un elemento de cada cinta
                elementos_por_cinta[robot->cinta1] --;
                elementos_por_cinta[robot->cinta2] --;

                // Avisa de que la cinta ya ninguna de las dos cintas
esta llena (ha consumido un elemento de cada una)
                pthread_cond_signal(&cinta_llena[robot->cinta1]);
                pthread_cond_signal(&cinta_llena[robot->cinta2]);

                printf("El robot %d duerme tras haber producido. \n",
robot->id);
                pthread_mutex_unlock(&mutex);

                sleep(DORMIR_ROBOT);
        }

        free(robot);

        pthread_exit(0);
}

/*
 * Lanza la producción en la fábrica.
 * Crea los hilos y espera por su finalización.
 */
void lanzar_fabrica(){
        struct datos_cinta **cinta;
        struct datos_robot **robot;
        // Dependencias entre cintas y robots
        int cinta_robot_1[] = {0,0,1};
        int cinta_robot_2[] = {0,1,0};
        //Dependencias entre robots y cintas
        int robot_cinta_1[] = {0,1};
        int robot_cinta_2[] = {1,2};

        // Se reserva espacio para todos los elementos
```

```
        threads=malloc(sizeof(pthread_t)*(NUM_CINTAS+NUM_ROBOTS));
        cinta_llena=malloc(sizeof(pthread_cond_t)*(NUM_CINTAS));
        cinta_vacia=malloc(sizeof(pthread_cond_t)*(NUM_CINTAS));
        elementos_por_cinta=malloc(sizeof(int)*(NUM_CINTAS));

        // Se inicializan los mecanismos de sincronizacion globales
        pthread_mutex_init(&mutex, NULL);
        pthread_barrier_init(&barrera, NULL, NUM_CINTAS+NUM_ROBOTS);
        int i;
        int j = 0;

        // Se inicializan las cintas
        cinta = (struct datos_cinta **)malloc(sizeof(struct datos_cinta
*)*NUM_CINTAS);
        for(i=0;i<NUM_CINTAS;i++){
                elementos_por_cinta[i] = 0;
                pthread_cond_init(&cinta_llena[i], NULL);
                pthread_cond_init(&cinta_vacia[i], NULL);

                cinta[i]=(struct datos_cinta *)malloc(sizeof(struct
datos_cinta));
                cinta[i]->id = i;
                cinta[i]->robot1 = cinta_robot_1[i];
                cinta[i]->robot2 = cinta_robot_2[i];
                if(pthread_create(&threads[j], NULL, hilo_cinta, (void
*)cinta[i])<0){
                        perror("No se puede crear el thread.");
                }
                j++;
        }

        // Se inicializan los robots
        robot = (struct datos_robot **)malloc(sizeof(struct datos_robot
*)*NUM_ROBOTS);
        for( i = 0; i < NUM_ROBOTS; i++){
                robot[i]=(struct datos_robot *)malloc(sizeof(struct
datos_robot));
                robot[i]->id = i;
                robot[i]->cinta1 = robot_cinta_1[i];
                robot[i]->cinta2 = robot_cinta_2[i];
                if(pthread_create(&threads[j], NULL, hilo_robot, (void
*)robot[i])<0){
                        perror("No se puede crear el thread.");
                }
                j++;
        }

        // Espera por la finalizacion de los hilos
        for(i=0;i<NUM_CINTAS+NUM_ROBOTS;i++){
                pthread_join(threads[i],NULL);
```

```
        }

        // Liberar recursos
        // ...

        for(i=0;i<NUM_CINTAS;i++){
                pthread_cond_destroy(&cinta_llena[i]);
                pthread_cond_destroy(&cinta_vacia[i]);
        }
}


/*
 * Launch the factory and wait for it to finish the production.
 */
int main (int argc, char ** argv){

        lanzar_fabrica();

        return 0;
}
```