

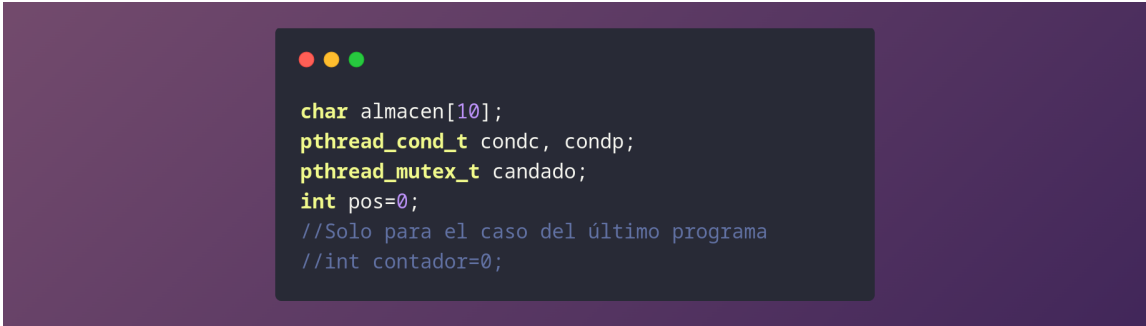
Reporte: Interbloqueos

Diego Ruiz Mora — 2202000335

9-Octubre-2023

1 Variables Globales

Todos los programas contienen un par de variables globales con el mismo propósito en todos los programas. Notaremos que solo el ultimo de los códigos usará la variable global que esta comentada, que funcionará para hacer el conteo de veces que se ejecutan los hilos. Encontraremos la variable

A screenshot of a code editor with a dark background and light-colored text. The code defines several global variables: a character array 'almacen' of size 10, two pthread condition variables 'condc' and 'condp', a pthread mutex 'candado', an integer 'pos' initialized to 0, and two commented-out integer variables 'contador' and 'contador'.

```
char almacen[10];
pthread_cond_t condc, condp;
pthread_mutex_t candado;
int pos=0;
//Solo para el caso del último programa
//int contador=0;
```

Imagen 1: Variables Globales

almacén del tipo arreglo de caracteres con una longitud de diez, que servirá para mostrar los productos generados o consumidos. Por otro lado tenemos *condc* y *condp* que son variables de condición que nos permite bloquear los hilos si cumplen ciertas condiciones o igual despertar estos hilos en caso de cumplir ciertas características. A su vez, tenemos también una variable del tipo mutex *candado* que nos ayudará a delimitar la región crítica y el acceso de cada uno de los hilos a la misma. Por último las variables enteras de *pos* que indica la posición dentro del almacén y el contador, que ya se mencionó su uso.


2 Funciones del programa

Muchas de las funciones de los códigos son iguales, algunas solo varían en una pequeña cosa, de manera conveniente explicaremos la función solamente una vez para ahorrarnos la necesidad de repetirlo en cada uno de los códigos.

2.1 Función Principal

Comenzamos por la función principal, donde encontramos la declaración de variables del tipo `pthread_t` que serán los dos hilos que utilizamos para consumir y producir elementos dentro del almacén. Para ello tenemos que inicializar las variables de condición para cada uno de los hilos, además hacemos la creación de los hilos dándoles como imagen la función *intentaRegionCritica*, recordemos que tenemos que esperar a que termine de ejecutar cada uno de los hilos creados.

Para finalizar destruimos las condiciones y el candado, esto con las ultimas tres instrucciones. donde pasamos por referencia los valores de los respectivos.

A screenshot of a code editor with a dark background and purple accents. The code is written in C and defines the `main` function. It includes the declaration of two threads `p` and `c` of type `pthread_t`. It then initializes a mutex `&candado` and two condition variables `&condc` and `&condp` using `pthread_mutex_init` and `pthread_cond_init` respectively. Two threads are created using `pthread_create`, both pointing to the `intentaRegionCritica` function. The first thread is given a size of 0 and the second a size of 1. After creating the threads, the program calls `pthread_join` to wait for both to finish. Finally, it destroys the mutex and condition variables using `pthread_mutex_destroy` and `pthread_cond_destroy`, and returns 0.

```
int main(){
    pthread_t p,c;
    pthread_mutex_init(&candado, NULL);
    pthread_cond_init(&condc, NULL);
    pthread_cond_init(&condp, NULL);
    pthread_create(&p,NULL, intentaRegionCritica, (void *)(&size_t)0);
    pthread_create(&c,NULL, intentaRegionCritica, (void *)(&size_t)1);
    pthread_join(p, NULL);
    pthread_join(c, NULL);
    pthread_cond_destroy(&condc);
    pthread_cond_destroy(&condp);
    pthread_mutex_destroy(&candado);
    return 0;
}
```

Imagen 2: Función Principal

2.2 Función para cada hilo

La llamada función *intentaRegionCritica* que es la imagen de cada uno de los hilos recibe como parámetro el número de hilo, que justamente nos funcionará para identificar que función particular ejecutará, si la de consumir o la de producir, pero para ello primero tenemos que generar un ciclo de cincuenta repeticiones y dentro de el haremos la delimitación de la región critica para que cada uno de los hilos puedan acceder de manera ordenada a ella. La delimitación laharemos abriendo y cerrando el mutex, todo esto dentro del ciclo, para que lo repita cincuenta veces por cada uno de los hilos, para darnos un total de cien veces.

```

void * intentaRegionCritica(void * arg){
    int id = (int) (size_t) arg;
    int i;
    for(i=0; i<50;i++){
        pthread_mutex_lock(&candado);
        if(id==0)
            produce();
        else
            consume();
        pthread_mutex_unlock(&candado);
    }
    pthread_exit(0);
}

```

Imagen 3: Función intentaRegionCritica

2.3 Función 'consume'

Esta función nos permitirá consumir los productos del almacén para ello utilizamos primero un condicional para el caso donde el la posición del almacén sea igual a cero, esto ocurre porque no tenemos elementos en el almacén, no tiene sentido que consumamos, por lo que se dormirá este proceso. Por otro lado , en cualquier otra posición se debería de consumir el producto, que en este caso lo hace asignando el carácter nulo y restando uno a posición, para notarlo imprimimos el *almacén* para notar las diferencias. Por ultimo despertamos al otro hilo para que pueda producir. Cabe notar que esta función solamente aplica a los dos primeros códigos, ya que en el tercero necesitaremos otras condicionales, que son muy parecidas.

```

int consume(){
    if(pos==0){
        pthread_cond_wait(&condc,&candado);
    }
    almacen[pos-1]='\0';
    pos--;
    puts(almacen);
    pthread_cond_signal(&condp);
}

```

Imagen 4: Función 'consume'