Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de Ciencias Computacionales

Ingeniería Informática

Asignatura**: Uso, Adaptación y Explotación de Sistemas Operativos**
Clave de Asignatura: **I5903**
NRC: **103860**
Sección: **D02**

Barbero Dormilón

Alumno: **Alejandro García Ventura**
Raymon Alejandro Ortiz Ríos
Juan Ricardo Gutiérrez Macias
Código: **219750329**
Profesor: **Violeta del Rocío Becerra Velázquez**
Fecha: **18/02/2022**

Tabla de contenido

Problema del barbero dormilón

Uno de los problemas clásicos para el cómputo paralelo es el barbero durmiente. Según narra la historia, tenemos una barbería, con una única silla y un único barbero, el cuál debe atender a todos los clientes que llegan. El problema consiste en un barbero durmiente, que siempre se duerme cuando no existen clientes en espera, los cuáles al llegar se sientan en una fila de sillas. Si un cliente llega y el barbero está durmiendo, éste lo despierta y lo comienza a afeitar, pero si se encuentra atendiendo a otro cliente, se queda esperando en la silla hasta que el barbero se desocupe.

En programación paralela, la zona crítica es aquel recurso compartido que sólo debe ser accedido por un solo hilo a la vez, para evitar la competencia por el recurso, y para ello el barbero durmiente ha sido uno de los problemas clásicos que utiliza mecanismos de exclusión mutua para resolver el problema de competencia por un recurso dentro de la región crítica.

El problema consiste en realizar la actividad del barbero sin que ocurran condiciones de carrera. La solución implica el uso de semáforos y objetos de exclusión mutua para proteger la sección crítica.

Un semáforo es una variable protegida (o tipo abstracto de datos) que constituye el método clásico para restringir o permitir el acceso a recursos compartidos (por ejemplo, un recurso de almacenamiento) en un entorno de multiprocesamiento. Fueron inventados por Edsger Dijkstra y se usaron por primera vez en el sistema operativo THEOS.

Código Fuente comentado

```
 1  #include<stdio.h>
 2  #include<stdlib.h>
 3  #include<unistd.h>
 4  #include <stdio.h>
 5  #include <unistd.h>
 6  #include <stdlib.h>
 7  #include <time.h>
 8  #include <pthread.h>
 9  #include <semaphore.h>
10
11  // El numero maximo de subprocesos de clientes
12  #define MAX_CUSTOMERS 25
13
14  // Prototipos de funcionesâ€¦
15  void *customer(void *num);
```

Alejandro García Ventura
Raymon Alejandro Ortiz Ríos
Juan Ricardo Gutiérrez Macias

```
16  void *barber(void *);
17
18  void randwait(int secs);
19
20  // Define los semaforos
21
22  // waitingRoom Limita el nÃºmero de clientes permitidos
23  // para entrar en la sala de espera a la vez
24  sem_t waitingRoom;
25
26  // barberChair garantiza acceso mutuamente exclusivo a la
silla de barbero
27  sem_t barberChair;
28
29  // barberPillow se utiliza para permitir que el peluquero
duerma hasta que llegue un cliente
30  sem_t barberPillow;
31
32  // seatBelt se utiliza para hacer que el cliente espere
hasta que el barbero termine de cortarle el cabello
33  sem_t seatBelt;
34
35  // Marcar para detener el hilo de barbero cuando todos los
clientes hayan sido atendidos
36  int allDone = 0;
37
38  int main(int argc, char *argv[]) {
39
40  pthread_t btid;
41  pthread_t tid[MAX_CUSTOMERS];
42  long RandSeed;
43  int i, numCustomers, numChairs;
44  int Number[MAX_CUSTOMERS];
45
46  printf("Ingrese el numero de Clientes : ");
scanf("%d",&numCustomers) ;
47  printf("Ingrese el numero de sillas : ");
scanf("%d",&numChairs);
48
49  // Asegura que la cantidad de subprocesos sea menor que la
cantidad de clientes que podemos atender
50  if (numCustomers > MAX_CUSTOMERS) {
51  printf("El numero maximo de Clientes es %d.\n",
MAX_CUSTOMERS);
52  exit(-1);
53  }
54
```

```
 55  // Inicializa el array de nÃºmeros
 56  for (i=0; i<MAX_CUSTOMERS; i++) {
 57  Number[i] = i;
 58  }
 59
 60  // Inicializa los semaforos con los valores iniciales
 61  sem_init(&waitingRoom, 0, numChairs);
 62  sem_init(&barberChair, 0, 1);
 63  sem_init(&barberPillow, 0, 0);
 64  sem_init(&seatBelt, 0, 0);
 65
 66  // Crea el barbero.
 67  pthread_create(&btid, NULL, barber, NULL);
 68
 69  // Crea los clientes.
 70  for (i=0; i<numCustomers; i++) {
 71  pthread_create(&tid[i], NULL, customer, (void
*)&Number[i]);
 72  sleep(1);
 73  }
 74
 75  // Une a cada uno de los hilos para esperar a que terminen
 76  for (i=0; i<numCustomers; i++) {
 77  pthread_join(tid[i],NULL);
 78  sleep(1);
 79  }
 80
 81  // Cuando todos los clientes hayan terminado, mata el hilo
del barbero
 82  allDone = 1;
 83  sem_post(&barberPillow); // Despierta al barbero para que
salga
 84  pthread_join(btid,NULL);
 85  }
 86
 87
 88  void *customer(void *number) {
 89  int num = *(int *)number;
 90
 91  // Sale para la tienda y toma una cantidad aleatoria de
tiempo para llegar
 92  printf("El cliente %d se va a la peluqueria.\n", num);
 93  randwait(2);
 94  printf("El cliente %d llegÃ³ a la peluqueria.\n", num);
 95
 96  // Espera a que se abra espacio en la sala de espera
 97  sem_wait(&waitingRoom);
```

```
 98  printf("Cliente %d entrando a la sala de espera.\n", num);
 99
100  // Espera a que la silla de barbero quede libre
101  sem_wait(&barberChair);
102
103  // La silla estÃ¡ libre asÃ- que cede su lugar en la sala
de espera
104  sem_post(&waitingRoom);
105
106  // Despierta al barbero
107  printf("Cliente %d despertando al barbero.\n", num);
108  sem_post(&barberPillow);
109
110  // Espera a que el barbero termine de cortarte el pelo
111  sem_wait(&seatBelt);
112
113  // abandona la silla
114  sem_post(&barberChair);
115  printf("Cliente %d saliendo de la peluqueria.\n", num);
116  }
117
118  void *barber(void *junk) {
119  // Si bien todavÃ-a hay clientes para ser atendidos
120  // Nuestro barbero es omnisciente y puede saber si
todavÃ-a hay clientes en camino a su tienda
121  while (!allDone) {
122
123  // Duerme hasta que alguien llegue y lo despierte
124  printf("El barbero esta durmiendo\n");
125  sem_wait(&barberPillow);
126
127  // Salta estas cosas al final
128  if (!allDone) {
129
130  // Toma una cantidad aleatoria de tiempo para cortar el
cabello del cliente
131  printf("El barbero esta cortando el cabello\n");
132  randwait(2);
133  printf("EL barbero ha terminado de cortar el cabello.\n");
134
135  // Suelta al cliente cuando termine de cortar
136  sem_post(&seatBelt);
137  }
138  else {
139  printf("El peluquero se va a casa por el dia.\n");
140  }
141  }
```

```
142  }
143
144  void randwait(int secs) {
145  int len;
146
147  // Generar un nÃºmero aleatorio
148  len = (int) ((1 * secs) + 1);
149  sleep(len);
150  }
```

Enlace al video del funcionamiento

https://youtu.be/MMa6JBAYAlg

Conclusión

En conclusión, el algoritmo del barbero dormilón sirve para resolver problemas de sincronización En realidad, este problema está basado en la vida real ya que es una cuestión de tiempo, esta comparación se hace para que siempre sepamos qué tipo de proceso tenemos que esperar a que termine un proceso para que el otro proceso comience a procesarse, o más bien, cada proceso. los procesos deben procesarlos en orden y permitir una cierta cantidad de tiempo para continuar.

Referencias

https://bitcu.co/barbero-durmiente/

http://diccionario.sensagent.com/Problema%20del%20barbero%20durmiente/es-es/

Alejandro García Ventura
Raymon Alejandro Ortiz Ríos
Juan Ricardo Gutiérrez Macias