SEMÁFOROS

GRUPO TERABYTES

Adryele Oliveira, Amanda Lopes, Livia Hipólito,
Filipe Côrrea, Felipe Alves

PROGRAMA EXPLICADO

Neste código usamos threads e semáforos para gerir a tráfego de clientes na barbearia.
Usamos 4 semáforos para controlar o acesso dos recursos.

```
7  #define NUM_CADEIRAS 20
8
9  sem_t clientes; // Semáforo para controlar os clientes na fila de espera
10 sem_t cadeira; // Semáforo para controlar a cadeira do barbeiro
11 sem_t travesseiro; // Semáforo para acordar o barbeiro
12 sem_t seatBelt; // Semáforo para sinalizar o término do corte de cabelo
```

PROGRAMA EXPLICADO

Adicionamos uma estrutura nomeada "Queue" para representar a fila de espera, uma função "Enqueue" para gerir a entrada de clientes na fila de espera e uma função "Dequeue" para gerir a saída de clientes da fila de espera.

```
19  struct Queue {
20      int data[NUM_CADEIRAS];
21      int front, rear;
22  } clienteQueue; // Estrutura de dados para representar a fila de espera dos clientes
23
24  void enqueue(int value) {
25      clienteQueue.rear = (clienteQueue.rear + 1) % NUM_CADEIRAS;
26      clienteQueue.data[clienteQueue.rear] = value;
27  }
28
29  int dequeue() {
30      int value = clienteQueue.data[clienteQueue.front];
31      clienteQueue.front = (clienteQueue.front + 1) % NUM_CADEIRAS;
32      return value;
33  }
```

PROGRAMA EXPLICADO

Então, cria-se a função denominada “cliente” para controlar o comportamento dos clientes através do uso dos semáforos que foram definidos no início do código, faz-se o mesmos para os outros elementos.

```
51  void *cliente(void *arg) {
52      int id = *((int *)arg);
53
54      // Tenta entrar na fila de espera
55      pthread_mutex_lock(&mutex);
56      if (clientesEsperando < NUM_CADEIRAS) {
57          clientesEsperando++;
58          printf("Cliente %d chegou.\n", id);
59          enqueue(id);
60          pthread_mutex_unlock(&mutex);
61
62          // Aguarda a vez na fila de espera
63          sem_wait(&cadeira);
64          printf("Cliente %d é o próximo da fila. \n", id);
65          sleep(1);
```

PROGRAMA EXPLICADO

Então, cria-se a função denominada “cliente” para controlar o comportamento dos clientes através do uso dos semáforos que foram definidos no início do código, faz-se o mesmos para os outros elementos.

```
67          // Cliente acordou o barbeiro e está cortando o cabelo
68          printf("Cliente %d acordou o barbeiro e está cortando o cabelo.\n", id);
69          sleep(2);
70
71          // Cliente terminou o corte de cabelo
72          printf("Cliente %d terminou o corte de cabelo e foi embora.\n", id);
73          sem_post(&clientes); // Libera o barbeiro
74          sem_post(&seatBelt); // Sinaliza o término do corte
75      } else {
76          // A fila de espera está cheia, o cliente vai embora insatisfeito
77          pthread_mutex_unlock(&mutex);
78          printf("Cliente %d viu o tamanho da fila e foi embora.\n", id);
79      }
80  }
```

PROGRAMA EXPLICADO

Abrimos a função “main” para dar início no processo dos semáforos e estruturas de dados, além de criarmos as Threads do barbeiro e do clientes para simular o atendimento e chegada/saída.

```
97      for (int i = 0; i < numClientes; i++) {
98          clientesArgs[i] = i;
99          pthread_create(&clienteThreads[i], NULL, cliente, &clientesArgs[i]);
100         sleep(1); // Cria clientes com um pequeno atraso
101     }
102
103     for (int i = 0; i < numClientes; i++) {
104         pthread_join(clienteThreads[i], NULL);
105     }
106
107     // Aguarda o barbeiro terminar
108     pthread_cancel(barbeiroThread);
109     pthread_join(barbeiroThread, NULL);
110     printf("encerrou o horário de atendimento e o barbeiro dormiu na cadeira. \n");
111
112     return 0;
113 }
```

CONCLUSÃO

O programa implementa o problema do "Barbeiro Dorminhoco", onde múltiplos clientes aguardam na fila para serem atendidos por um único barbeiro. A utilização de semáforos e mutexes garante que o comportamento das threads seja coordenado de forma segura, respeitando as regras do problema.