这段代码实现了理发店（barbershop）问题，这是一个经典的并发控制问题，通常用于演示信号量的使用。该问题涉及一个理发师（barber）和多个顾客（customers），其中理发师为等待的顾客理发。代码使用 C 语言编写，并利用信号量来控制同步。

变量和信号量定义

c 复制代码

```c
int waiting = 0;           // 当前等待的顾客数量
int CHAIRS = N;            // 理发店里的椅子数量（即最大等待顾客数）
semaphore customers = 0;   // 等待理发的顾客数量
semaphore barbers = 0;     // 可用的理发师数量
semaphore mutex = 1;       // 互斥锁，用于保护对 waiting 变量的访问
```

理发师进程

c 复制代码

```c
process barber() {
    while (true) {
        P(customers);    // 等待有顾客
        P(mutex);        // 进入临界区
        waiting--;       // 一个顾客将要被服务，等待的顾客数减少
        V(barbers);      // 通知一个理发师可以理发了
        V(mutex);        // 退出临界区
        cut_hair();      // 理发（服务顾客）
    }
}
```

- **P(customers)**：理发师等待有顾客。如果没有顾客，理发师将阻塞在这里。
- **P(mutex)**：进入临界区，保护 `waiting` 变量。
- **waiting--**：减少等待的顾客数。
- **V(barbers)**：通知一个理发师可以理发了。
- **V(mutex)**：退出临界区，释放对 `waiting` 变量的保护。
- **cut_hair()**：进行理发操作。

顾客进程

```c
process customer_i() {
    P(mutex);                   // 进入临界区
    if (waiting < CHAIRS) {     // 如果有空的椅子
        waiting++;              // 增加等待的顾客数
        V(customers);           // 通知理发师有顾客等待
        V(mutex);               // 退出临界区
        P(barbers);             // 等待理发师
        get_haircut();          // 接受理发服务
    } else {                    // 如果没有空的椅子
        V(mutex);               // 退出临界区
    }
}
```

- **P(mutex)**：进入临界区，保护 `waiting` 变量。
- **if (waiting < CHAIRS)**：检查是否有空的椅子。
  - **waiting++**：增加等待的顾客数。
  - **V(customers)**：通知理发师有顾客等待。
  - **V(mutex)**：退出临界区，释放对 `waiting` 变量的保护。
  - **P(barbers)**：等待理发师准备好理发。
  - **get_haircut()**：接受理发服务。
- **else**：如果没有空的椅子，顾客离开。
  - **V(mutex)**：退出临界区。

总结

- **理发师进程**：理发师一直在等待有顾客，如果有顾客，理发师会理发并减少等待的顾客数。
- **顾客进程**：顾客到达时，如果有空的椅子，顾客会坐下并等待理发师；如果没有空的椅子，顾客会离开。