> 详细分析Peterson算法如何实现进程互斥，并且分析方法巧妙之处。

Peterson算法：

```
#define FALSE 0
#define TRUE 1
#define N 2  // 进程的个数
int turn;
int interested[N];

void enter_region(int process) {
    int other;
    other = 1 - process; // 另一个进程的进程号
    interested[process] = TRUE; // 表明本进程感兴趣
    turn = process; // 设置标志位

    while(turn == process && interested[other] == TRUE);
}

void leave_region(int process) {
    interested[process] = FALSE; // 本进程已离开临界区
}
```

Peterson算法使用两个变量 `interested` 与 `turn` 实现进程互斥，当 `interested[N]` 的值为真，表示进程号为 `N` 的进程希望进入该临界区，变量 `turn` 保存有权访问共享资源的进程的进程号。

互斥：当进程0在临界区时， `interested[0]` 为真，进程1在临界区外等待。当进程1在临界区时同样。

巧妙之处：通过 `turn` 解决了只有 `interested[N]` 会导致**死锁**的问题，再结合 `interested[N]` 解决只有 `turn` 时只能交替进入的临界区的问题。满足progress原则。

> 如何使用信号量方法实现屏障（Barrier）？

Barrier实验：

```
int n; // 进程的数量
int count = 0; // 记录到达汇合点的进程数
mutex = Semaphore(1); // 保护count
barrier = Semaphore(0); // 进程都达到前小于等于0，达到后取正值


void Barrier(...){
    /*Barrier*/
    wait(&mutex);
    count = count + 1;
    signal(&mutex);
    if(count == n) {
        count = 0;
        for(j = 0; j < n; ++j)
            signal(&barrier);
    }else {
```

```
        wait(&barrier);
    }
}
```

每当进程到达时，通过互斥记录 `count` 的值，当所有进程都达到汇合点时，唤起所有进程；否则阻塞，最终实现屏障（Barrier）。