南 开 大 学操作系统期末论文

中文题目：多个理发师问题设计报告

学 号： 2013852

姓 名： 郁万祥

年 级： 2020 级

专 业： 软件工程

学 院： 软件学院

指导教师： 李旭东

完成日期： 2022 年 12 月 30 日

目　录

中文题目：多个理发师问题设计报告 1
一、 问题描述 3
1.1 理发师问题 3
1.2 多个理发师问题 3
1.3 实际问题描述 3
二、 算法原理 5
2.1 基本思想 5
2.2 相关调用 5
2.3 算法原理 6
三、 具体实现 8
3.1 数据结构 8
3.2 理发师进程 8
3.3 顾客进程 9
四、 测试数据 10
4.1 barber_test1 10
4.2 barber_test2 10
4.3 barber_test3 10
五、 思考总结 11

一、问题描述

1.1 理发师问题

一个理发店由一个有几张椅子的等待室和一个放有一张理发椅的理发室组成。

1、若没有要理发的顾客，则理发师去睡觉：

2、若一顾客进入理发店，理发师正在为别人理发，且等待室有空椅子，则该顾客就找张椅子按顺序坐下：

3、若一顾客进入理发店，理发师在睡觉，则叫醒理发师为该顾客理发：

4、若一顾客进入理发店且所有椅子都被占用了，则该顾客就离开。

这个问题当中，只有一个理发师进程。

1.2 多个理发师问题

对于上述一个理发师问题，我们将问题进行拓展：

一个理发店由一个有 n 个椅子的等候室和一个有 m 个理发椅（理发师）的理发室组成。

1、如果有没有顾客可以服务，所有的理发师都去睡觉。

2、如果顾客走进理发店椅子被占用了，然后顾客离开了商店。

3、如果所有的理发师都很忙，但是椅子是可用的，然后顾客坐在一张免费的椅子上。

4、如果理发师睡着了，顾客就会叫醒理发师。

此时问题当中就有多个理发师的进程。

1.3 实际问题描述

针对于现实情况，我们继续将问题进行扩展。

1、理发店有 n 把椅子，n 位理发师，一个等待区可供 m 位顾客在沙发上等待其他的顾客可以站着等待，但出于消防安全考虑，要求理发店中总顾客数目不超过 M 人。

2、如果理发店已经满了，顾客不会进来。

3、如果进来，首先选择沙发，然后选择站着。

4、理发师空闲时，向坐在沙发上的顾客提供服务。

5、如果有站着的用户，来店时间最长的顾客坐到沙发上。

6、顾客理发结束，任何理发师可以收钱，但只有一个收款机，一次只能接受一位顾客的付款。

此次期末实践，将对这个问题进行模拟。

二、算法原理

2.1 基本思想

为了实现多个理发师问题，需要将通过信号量的设置，从而解决阻塞问题，实现多个理发师共享顾客队列。

1) 、信号量 customers 用来记录等候理发的顾客数，并用作阻塞理发师进程，初值为 0：

2) 、信号量 barbers 用来记录正在等候顾客的理发师数，并用作阻塞顾客进程，初值为 0(刚开始时理发师在睡觉，所以理发师这个资源数目为 0):

3) 、信号量 mutex 用于互斥，初值为 1.

4）、信号量 pay、over、cash_r 用来记录正在付款，作用是阻塞其他付款的顾客进程。

5）、信号量 wanttosit 用来记录沙发上的等待顾客数，问题要求只有 4 个沙发共休息，其他人需要站着等待，用于阻塞站着的顾客到沙发休息。

2.2 相关调用

1）、Linux 信号量工具：

include<semaphore.h>中定义了很多信息量操作中常用的数据结构和系统函数，下面罗列了本次实验将用到的：

sem_t:具体信号量的数据结构

sem_init：用于创建信号量，并能初始化它的值

sem_wait:相当于 wait 操作

sem_post:相当于 signal 操作

2)、POSIX 线程相关：

#include<pthread.h>中用到的数据类型和函数如：

pthread_t:用于声明线程 ID

pthread_create:创建一个线程

pthread_join:阻塞当前线程，直到另外一个线程运行结束

3) 、常用标准库等

2.3 算法原理

1）、所有理发师使用相同的程序段。

2）、所有顾客使用相同的程序段。

3）、竞争资源：理发师、顾客

4）、同步关系：理发师和顾客是同步关系：理发师等待顾客来，然后为顾客服务，顾客来了之后叫醒理发师，执行上是有先后顺序的，所以应该有两个同步信号量，目散在两个进程里：同步关系是对称的。

5）、互斥关系：

顾客之间有互斥关系（对理发师的访问，一个理发师一次只能为一个顾客服务，对付款及的访问，付款机一次只能为一个顾客服务）。

顾客和理发师之间有互斥关系（访问（读/写）顾客计数变量)。

理发师之间有互斥关系（对顾客的访问，一个顾客一次只能被一个理发师服务）。

6）、顾客行为：

顾客到店，互斥地访问/检查是否有等待沙发可以入座

试图坐下等待（互斥的访问已有顾客数量，并与沙发总数比较）

case1：如果有空位：

试图唤醒理发师，告诉理发师，现在顾客来啦；（同步关系）

互斥地统计变量；（等待理发的顾客数+1）

等待理发师准备完毕来为他执行理发；

享受理发过程；

case1：如果有人正在付款，等待。

case2：如果没有人正在付款：

进行付款；

完事后顾客线程结束；

case2：如果没有空位：

释放掉互斥锁直接离开。

7）、理发师行为：

理发师工作是一个循环；

互斥的检查理发的顾客数量；

case1：如果发现没有等待的顾客，理发师睡眠。

case2：如果发现有等待服务的顾客：

顾客通过 wait(barbers)来唤醒阻塞的理发师；

互斥的将等待的顾客数量-1；

模拟理发流程；

case1：如果有人正在付款，进行等待。

case2：如果没有人正在付款：

通过 wait(pay)唤醒阻塞的付款机；

模拟付款流程；

进入理发师线程循环的下一轮；

三、具体实现

3.1 数据结构

```
sem_t customers;  // 是否有顾客，理发师和顾客之间的同步信号量
sem_t barbers;  // 理发师状态，用于顾客互斥使用理发师资源
sem_t mutex; // 理发师之间的互斥锁
sem_t wanttosit;  // 是否有沙发等待，用于顾客互斥使用沙发资源
sem_t pay;  // 是否有人付款，用于顾客互斥使用付款资源
sem_t over;  // 理发是否结束，用于进入到付款阶段
sem_t cash_r;  // 是否有人付款，用于理发师互斥使用付款资源
int waiting=0;   //  等待顾客的数目，用于判断是否达到店内最多容纳顾客数目
```

3.2 理发师进程

```
void *barber_1(void*arg)
{
   while(1){
   sem_wait(&customers);
   sem_wait(&mutex);
   sem_post(&barbers);
   sem_post(&mutex);
   printf("2013852ywx的理发店：1号理发师开始理发\n");
   sleep(10);
   sem_post(&over);
   sem_wait(&pay);
   sem_wait(&cash_r);
   printf("2013852ywx的理发店：1号理发师为客人结帐\n");
   sem_post(&cash_r);
   waiting--;

   }
}
```

3.3 顾客进程

```
void *customer_i(int b)
{
  printf("%d号客人来了!\n",b+1);
  sem_wait(&mutex);
  if(waiting<20)
 {
     waiting++;
     sem_post(&mutex);
     sem_post(&customers);
     switch(waiting){
     case 0 ... 3:
    {
     sem_wait(&barbers);
     printf("%d号客人正在理发\n",b+1);
     sem_wait(&over);
     printf("%d号客人去结帐\n",b+1);
     sem_post(&pay);
     break;
    }
     case 4 ... 7:
      {
        sem_wait(&wanttosit);
        printf("%d号客人去沙发等待\n",b+1);
       sem_wait(&barbers);
       sem_post(&wanttosit);
       printf("%d号客人正在理发\n",b+1);
       sem_wait(&over);
       printf("%d号客人去结帐\n",b+1);
       sem_post(&pay);
      break;
      }
     default :
      {
      printf("%d号客人站着等待\n",b+1);
       sem_wait(&wanttosit);
      printf("%d号客人去沙发等待\n",b+1);
      sem_wait(&barbers);
      sem_post(&wanttosit);
      printf("%d号客人正在理发\n",b+1);
      sem_wait(&over);
      printf("%d号客人去结帐\n",b+1);
      sem_post(&pay);
      break;
     }
     }
     }
     else
      {
      printf("无位置，%d号客人离开\n",b+1);
       sem_post(&mutex);

       }
}
```

四、测试数据

4.1 baber_test1:

该测试中，理发店最多容纳 20 人，一共有 3 个理发师，4 个可以坐的沙发（其他等待的顾客需要站着等待），总共来 20 名顾客。

4.2 barber_test2:

该测试中，理发店最多容纳 20 人，一共有 5 个理发师，4 个可以坐的沙发（其他等待的顾客需要站着等待），总共来 40 名顾客。

4.3 baber_test3:

该测试中，理发店最多容纳 10 人，一共有 5 个理发师，4 个可以坐的沙发（其他等待的顾客需要站着等待），总共来 40 名顾客。

具体测试运行结果截图见文件夹：实验结果

五、思考总结

作为操作系统期末的实践设计，我选择这个题目的原因，主要是考虑到在平时的实验课的过程中，就对于多线程的问题有所困惑，特别是对于使用信号量进行同步、互斥等操作的实操，当时在进行生产者消费者问题的时候就只是一知半解。然而，通过这一个学期的接触，包括在并行课程的学习当中，我发现对于多线程的操作还是十分重要的，因此，想通过这个机会将多线程的一些问题进行巩固以及再学习。

经过此次期末实践的设计，我意识到，多线程的同步互斥问题，在解决过程中，理清算法的流程是十分重要的，这样有助于在编程的时候理解到底什么时候进行 sem_wait，什么时候进行 sem_post，以及该对哪个信号量进行操作，就像这个多理发师问题一样，经过上述 3.3 算法原理的分析，理清了竞争资源的实体是顾客和理发师（这个比较容易分析）后，判断哪些关系存在同步，哪些关系存在互斥（这一步也比较容易，通常根据问题实际情况而定），然后关键的就是对于竞争实体的具体行为的分析，这时候需要考虑到哪些行为需要唤起阻塞，哪些行为的结束需要进行等待，哪些行为的结束不需要等待。再分析出来这些以后，在进行编程的时候，就像是使用编程语言对流水线进行翻译一样，就会变得比较清晰简答了。