Linux 系统分析实验报告

用信号量解决理发师问题

061261008 蒋炎岩 (一班)

1 实验要求

理发师问题：理发店理有一位理发师、一把理发椅和 5 把供等候理发的顾客坐的椅子。如果没有顾客，理发师便在理发椅上睡觉一个顾客到来时，它必须叫醒理发师，如果理发师正在理发时又有顾客来到，则如果有空椅子可坐，就坐下来等待，否则就离开。

用 Linux 线程机制和信号量机制解决这个问题，并且：

(1) 每个顾客进入理发室后，即时显示“Entered” 及其线程标识，还同时显示理发室共有几名顾客及其所坐的位置

(2) 至少有 10 个顾客，每人理发至少 3 秒钟。

(3) 多个顾客须共享操作函数代码。

2 背景知识

2.1 POSIX 线程

在一个程序中的多个执行路线称之为线程。Linux 在 1996 年第一次获得线程支持，现在已经有一套完整的与线程有关的函数库调用，大多数以 pthread_开头，编译时需要用 -lpthread 选项进行连接。

我们用函数 pthread_create 创建一个新线程：

```
#include <pthread.h>
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
void *(*start_routine)(void *), void *arg);
```

thread 参数表示新线程的标识符；attr 用于设置线程属性，如不需要设置，可设置为 NULL；start_routine 标识了线程启动程序的入口，arg 为传入的参数。调用 pthread_create 就可以立即创建一个新的执行路线，与原有线程共享所有的主存空间，但拥有独立的堆栈。

2.2 信号量

信号量通过一个计数器控制对共享资源的访问。

如果计数器大于 0，则访问被允许，如果为 0，则访问被禁止。计数器计算的结果是允许访问共享资源的通行证。因此，为了访问共享资源，线程必须从信号量得到通行证（P 操作），如果该信号量的计数大于 0，则此线程获得一个通行证，这将导致信号量的计数递减，否则，此线程将阻塞直到获得一个通行证为止。

当此线程不再需要访问共享资源时，它释放该通行证（V 操作），这导致信号量的计数递增，如果另一个线程等待通行证，则那个线程将在那时获得通行证。

信号量用于并发进程（线程）的同步、互斥。

2.3 线程信号量

Linux 为线程提供了一套不同于 System V IPC 的信号量机制。由于线程共享存储区的特殊性，这种信号量实现的机制比 IPC 中的信号量简单得多，无需陷入内核访问，效率也高得多，semaphore.h 定义了信号量结构 struct sem，通过 sem_init 函数，我们可以初始化信号

量：

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshard, unsigned int value);
```

然后，提供了 wait 和 post 两种操作：

```
int sem_wait(sem_t * sem);
int sem_post(sem_t * sem);
```

其中 wait 操作对应了传统信号量的 P 操作，post 操作对应了传统信号量的 V 操作。所以，我们还额外地定义了两个宏，使得我们的程序容易理解：

```
#define P(a) sem_wait(&a)
#define V(a) sem_post(&a)
```

这样，对于 sem 类型变量，就可以只写编写 P(mutex); V(mutex);这样的代码了。

3 问题分析与求解

3.1 全局变量与信号量设置

我们必须要为当前的座位情况设置一个计数器 nr_customer，表示当前坐在座位上等待的数量。同时，为了保证多个线程读写这个变量不会产生问题，需要设置一个信号量 mutex 用于全局的互斥。这个信号量除了用于进行 nr_customer 的互斥之外，还用来作为其他共享变量的互斥锁。

然后，我们设置全局数组 chair_status[NR_CUSTOMER]，来描述每一个座椅上的用户。这个数组的访问也需要借助 mutex 信号量的互斥。

另外，我们设置变量 current_barber，表示当前理发师正在服务的用户，-1 表示理发师空闲。设置这个变量是为了方便地进行图形输出。

我们设置信号量 customer，表示正在等待顾客的数量，它的初始值为 0，理发师线程通过 P(customer)来获得最终的理发权；设置信号量 barber，用于标识可用的理发师。理发者通过 P(barber)和 customer 信号量协作来完成整个理发开始前的同步过程。

最后，设置信号量 finish，用于理发结束后的同步。由于之前已经建立了线程的同步协作，此处只要一个信号量就可以解决问题，客户最终 V(finish)，在理发师处 P(finish)完成整个理发过程。

3.2 理发师线程

理发师线程执行顺序如下：

```
void *barber_thread(void *) {
    while (1) {
        P(customer); // 必须有顾客来到
        P(mutex); // 互斥锁 [1]
            nr_customer --;
            V(barber); // 理发师设置为可用
        V(mutex); // 解除互斥锁
        P(finish); // 完成同步，等待理发完成
        current_barber = -1; // 设置当前理发师状态
    }
}
```

理发师线程首先通过 P(customer)确认当前有顾客到来。从[1]开始之后，就保证了一定

存在阻塞的顾客线程。然后，理发师线程调用 V(barber)来确认自己的可用状态，再释放互斥锁后，P(finish)阻塞自己，此时必然是顾客线程开始执行，达到完整的同步目的。

3.3 顾客线程

顾客线程的执行顺序如下：

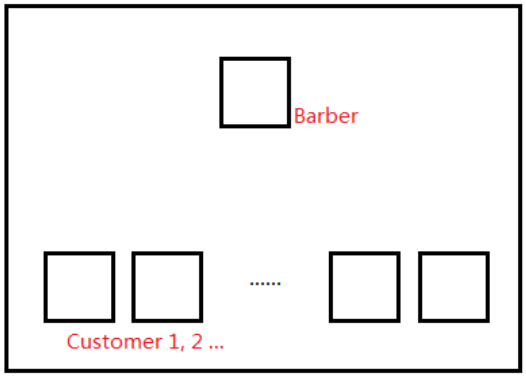
```
void *customer_thread(void *) {
    P(mutex); // 获得互斥锁
    if (nr_customer < NR_CUSTOMER) { // 保证座椅没有被坐满
        nr_customer ++; // 进入一个顾客
        V(customer); // 通知理发师线程有顾客到来
        V(mutex); // 释放互斥锁
        P(barber); // 试图获取理发师，无法获得的线程将阻塞
        P(mutex); // 进入临界区需要加锁
            current_barber = chair_status[i];
            chair_status[i] = -1;
        V(mutex);
        sleep(rand() % 4 + 2); // 理发过程
        V(finish); // 完成同步
    } else { // 座椅被坐满
        V(mutex); // 顾客离开
    }
}
```

顾客线程首先尝试获取互斥锁，然后读取当前座椅上的人数。如果已经坐满，就离开，执行 else 部分的代码，否则就进入服务状态。首先在互斥锁锁定的状态更新顾客信息，然后用 V(customer)告知理发师线程有顾客到来，并释放互斥锁。

执行完这部分代码后，就进入理发阶段，首先 P(barber)获得理发师，然后执行理发的过程。理发过程中，理发师一定是阻塞在 finish 上的，于是在理发完毕后，执行 V(finish)完成整个理发过程的同步。

3.4 主进程与图形界面

在主进程启动后，我们首先创建一个新线程，负责理发师问题的求解。然后，原有的执行路线创建 QApplication 窗口，并执行，主窗口结构如下：



在主窗口中，我们创建 N 个 QLabel 对象，并将初始图像修改为椅子的图片，理发师修改为睡觉的图形，然后设置一个定时器为 100ms，定时更新。

每当定时器到期时，我们都刷新当前的图形状态。但这需要在互斥的情况下执行，我们为主窗口的 timerEvent 编写代码：

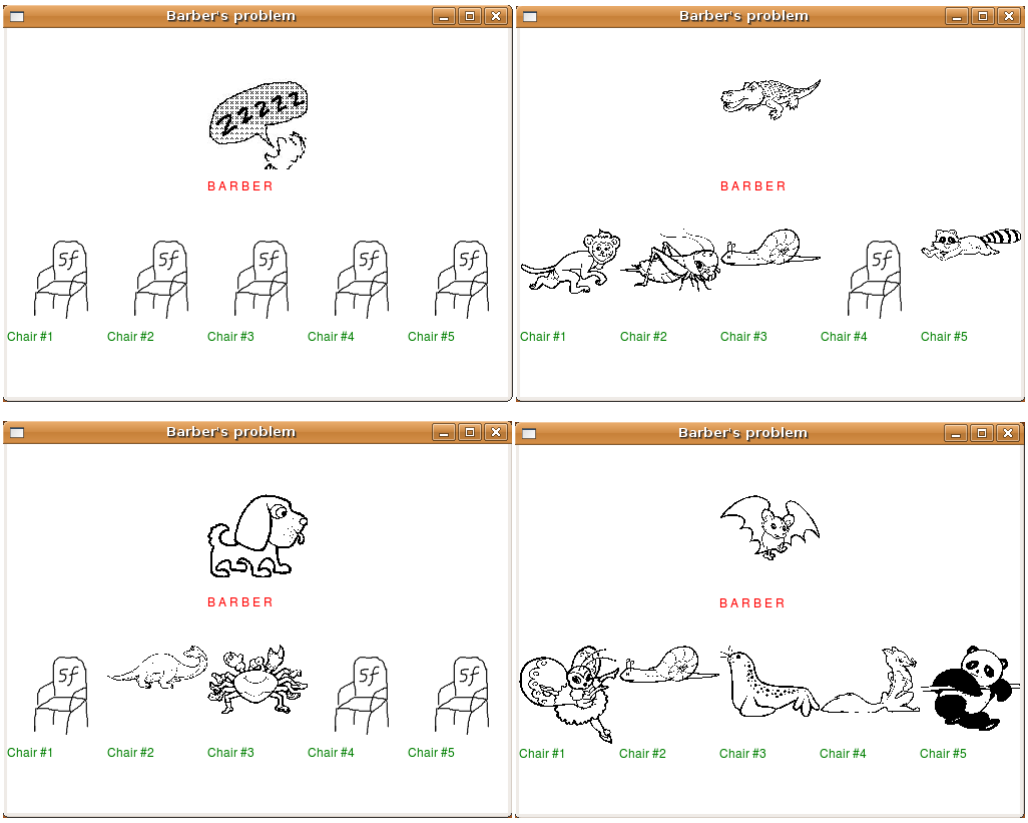
```
P(mutex);
for (i = 0; i < NR_CUSTOMER; i ++)
    data[i] = chair_status[i];
cnt = current_barber;
V(mutex);
```

将临界区的数值复制出来，然后重新绘制。

主进程则并发进行，启动理发师线程，并定期随机地启动一个新的顾客线程，达到演示的目的。

4 运行结果

运行程序：



左上图为初始状态，理发师处于睡觉状态，所有椅子都空闲。

程序执行了一段时间后，椅子将不再空闲，根据线程到来的顺序被信号量排队，于是产生了各种不同的中间结果。

左下图为全满的情况，此时再到来的客户将主动离开。

5 实验总结

1. 实践了 pthread 线程库的用法，以及不同于 System V IPC 中信号量的线程间通信机制。它比 IPC 中提供的机制效率更高、使用更简便，而且达到的功能相同，这是因为它的通信范围仅仅局限于同一个进程中的线程，而 IPC 中的信号量可以在任意进程之间进行同步。
2. 实践了信号量的应用。

3. 编写了 GUI 程序描述了理发师问题，可以很清晰地看到理发师问题的解决过程，同时也在控制台上有整个过程的输出。

6 代码清单

代码结构

src/common.h：公共头文件定义

src/glo.h：全局变量声明

src/window.h：程序窗口类定义

src/main.cpp：主程序文件，包括两个理发师和顾客线程

src/glo.cpp：全局变量定义

src/window.cpp：GUI 实现

编译方法

执行 make 即可编译。

依赖的库

libqt4-gui