实验一：进程间同步/互斥问题

wangqr

0. 问题描述

银行有 n 个柜员负责为顾客服务，顾客进入银行先取一个号码，然后等着叫号。当某个柜员空闲下来，就叫下一个号。

编程实现该问题，用 P、V 操作实现柜员和顾客的同步。

实现要求：

1. 某个号码只能由一名顾客取得；
2. 不能有多于一个柜员叫同一个号；
3. 有顾客的时候，柜员才叫号；
4. 无柜员空闲的时候，顾客需要等待；
5. 无顾客的时候，柜员需要等待。

1. 实现方式

此题最朴素的实现方法，是将每个柜员和顾客实现为一个线程。柜员线程负责不断叫号、服务顾客；而顾客线程仅负责在指定的进入时间去取号码。

然而注意到顾客线程的功能过于简单，我们可以不使用顾客线程，直接在主线程中按时间模拟顾客取号，而只保留柜员线程。

程序提供一个必须互斥访问的叫号队列，来完成线程间同步。叫号队列中的等候人数以一个信号量来表示。主线程按时向叫号队列中插入顾客，并升信号量。当柜员空闲时，尝试降信号量，成功后从叫号队列中叫号并服务。

在测试数据和柜员人数为 2 的条件下，时序图如图 1 所示。

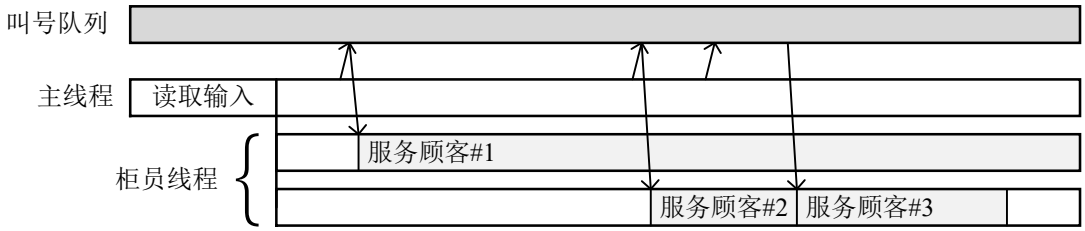

图 1 程序时序图

2. 代码实现

使用两个结构体来表示顾客和柜员。顾客结构体中包括顾客编号、所需时间、进入时间等数据；柜员结构体包括柜员的编号，和柜员的线程。

```
struct Customer {
    unsigned n, t_need;
    int t_in;
} *customers;

struct TellerThread {
```

```
    pthread_t p;
    unsigned n;
} *tellers;
```

线程间同步利用如下的一个信号量和一个互斥锁完成。

```
sem_t sem_customer;
pthread_mutex_t mutex_waiting_queue;
```

柜员线程的传入参数用来传入柜员结构体，返回值没有使用。柜员在一个循环中不断从去叫号并服务（sleep）顾客。由于需要一种机制在所有顾客服务完之后告知柜员结构体退出，因此主线程在插入最后一个顾客后，还会将信号量升至柜员数目，这样当柜员降信号量成功，但队列中无顾客时，柜员即可退出。

```
void *teller(void *param) {
    struct TellerThread *p = (struct TellerThread *) param;
    struct Customer *serving;
    unsigned called_num;
    while (1) {
        sem_wait(&sem_customer);
        pthread_mutex_lock(&mutex_waiting_queue);
        if (customers_waiting == customers_top) {
            pthread_mutex_unlock(&mutex_waiting_queue);
            break;
        }
        called_num = customers_waiting;
        serving = &customers[customers_waiting++];
        pthread_mutex_unlock(&mutex_waiting_queue);
      printf("[%.2f] Teller #%u started to serve customer #%u with number %u.\n",
               current_time(), p->n, serving->n, called_num);
        usleep(1000000u * serving->t_need);
      printf("[%.2f] Teller #%u finished serving customer #%u with number %u.\n",
               current_time(), p->n, serving->n, called_num);
    }
    return NULL;
}
```

完整的代码在 main.c 中，同一文件夹下的 demo.sh 提供了从编译至程序运行的完整演示。

3. 运行结果

在如下的系统环境中对所编写的程序进行了测试：

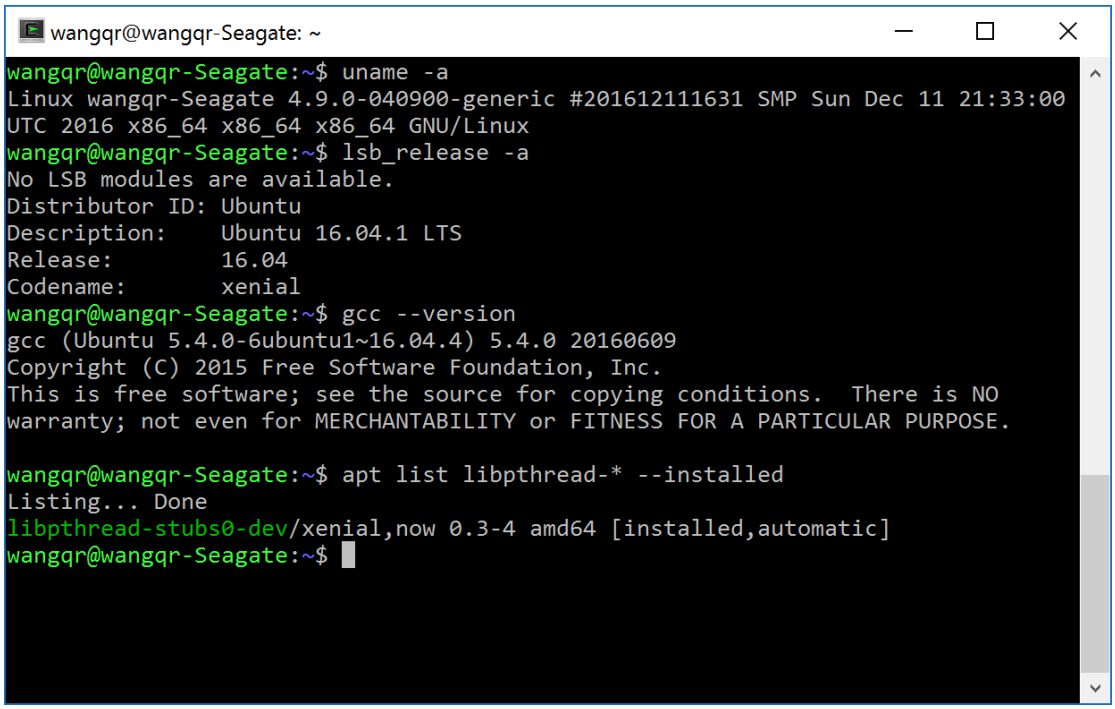

图 2 测试机器系统环境

测试结果如下图，其中，标有“Simulating sample case”的下方是 main.c 的运行输出。

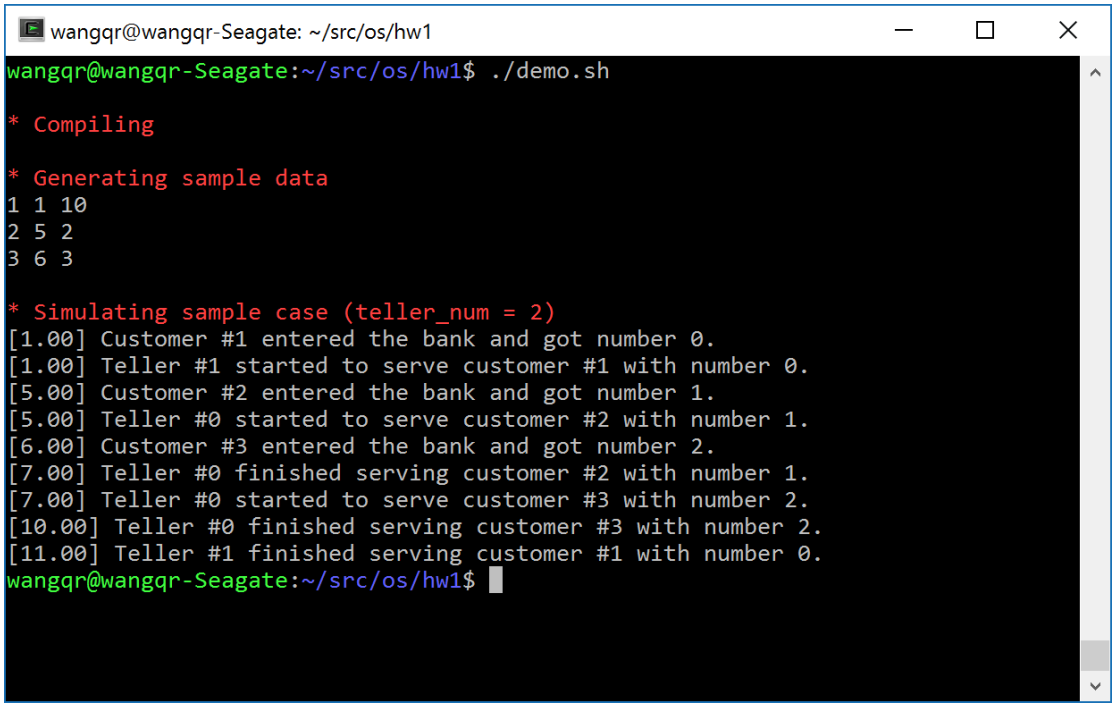

图 3 运行结果

从运行结果可以看到，仿真结果与图 1 一致，图 1 中的上方柜员线程对应仿真结果的 Teller #1，下方柜员线程对应仿真结果的 Teller #0，这在每次运行中是可能变化的。

4. 思考题

1. 柜员人数和顾客人数对结果分别有什么影响？

若顾客人数远多于柜员时，柜员基本不等待，则由于顾客所需服务时间总量一定，更多的柜员人数可以减少顾客的等待时间和服务所有顾客的总时间，服务所有顾客的总时间反比于柜员人数。当柜员数目与顾客人数相近时，大部分顾客都能够及时得到服务，服务所有顾客的总时间近似取决于到达时间加服务所需时间最晚的那一位客户。

2. 实现互斥的方法有哪些？各自有什么特点？效率如何？

常用的互斥方法如自旋锁、互斥锁、信号量。

自旋锁用于预先判定所需等待时间很短的加锁情形，但它将占满 CPU。

互斥锁最为常见，相比信号量开销较小，能够实现互斥。

信号量可以实现更加复杂的的互斥访问，比如当总资源数大于 1 时的场景。只使用 0/1 取值的信号量的功能与互斥锁较为类似，但由于互斥量中的等待队列的存在，使用信号量可以保证先阻塞的线程一定可以先获得资源，在某些场景下可以使调度比使用互斥锁更加公平。