操作系统第一组实验报告

漆耘含

2016011058

1　银行柜员服务问题

1.1　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。

1.2　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。

1.3　实现提示

1. 互斥对象：顾客拿号，柜员叫号；
2. 同步对象：顾客和柜员；
3. 等待同步对象的队列：等待的顾客，等待的柜员；
4. 所有数据结构在访问时也需要互斥。

1.4 测试文本格式

测试文件由若干记录组成，记录的字段用空格分开。记录第一个字段是顾客序号，第二字段为顾客进入银行的时间，第三字段是顾客需要服务的时间。

下面是一个测试数据文件的例子：

1 1 10

2 5 2

3 6 3

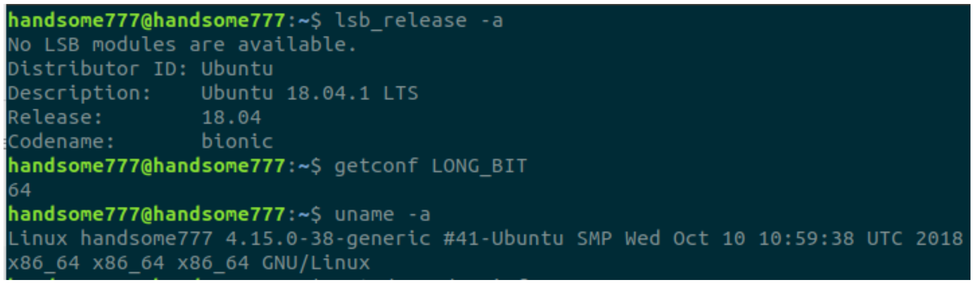
1.5 输出要求

对于每个顾客需输出进入银行的时间、开始服务的时间、离开银行的时间和服务柜员号。

2 实验环境

Ubuntu 18.04.1 LTS, 64位操作系统

具体配置如下：



```
handsome777@handsome777:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:        18.04
Codename:       bionic
handsome777@handsome777:~$ getconf LONG_BIT
64
handsome777@handsome777:~$ uname -a
Linux handsome777 4.15.0-38-generic #41-Ubuntu SMP Wed Oct 10 10:59:38 UTC 2018
x86_64 x86_64 x86_64 GNU/Linux
```

3 原理分析

针对这个银行柜台这个问题，先分清有两个对象，一个是柜台(Counter)，另一个是消费者(Customer)。下面就分别对柜台线程和消费者线程进行解析。

3.1 柜台

每一个柜台之间是相互独立的，所以刚开始应该为每一个柜台创建一个进程，以保证独立性。同时，柜台还需要编号、锁变量、进程变量等，因此柜台被封装为一个结构体，结构体的设计如下：

Algorithm 1 struct Counter

```
$pthread\_t\ c\_pthread_t$;//pthread var
$pthread\_mutex\_t\ c\_mutex\_t$;//mutex var
$pthread\_cond\_t\ c\_cond\_t$;//cond var
```

在柜台和消费者之外，有一个队列(queue)，里面是消费者按时间进入的顺序，先进先出，后进后出。在柜台流程刚开始的时候，先判断取出一个资源后，信号量是否小于等于0，如果小于等于0，则陷入阻塞，直到有消费者进入增加资源唤醒柜台，这时柜台进程才继续运行。

因为柜台进程之间是相互独立的，但他们都是从同一个队列(queue)里面读取消费者，因此可能被同时取出去，所以要加一个判断，即取出的消费者是否有对应的柜台号(消费者结构体定义见下一个小节)，如果有，则继续取或者阻塞，如果没有，则柜台与消费者匹配成功，然后进入服务，在这个时候要及时修改消费者的对应柜台编号，避免被同时服务的可能，柜台会陷入等待状态($pthread\_cond\_wait$)，直到消费者服务完成(sleep)，发送一个signal信号，唤醒柜台，然后结束服务。

值得注意的是，在所有消费者完成服务之后，所有的柜台进程都会陷入阻塞中，程序无法正常执行，因此需要设立一个全局变量($finish\_cus\_num$)，在主函数中会陷入忙等状态($pthread\_cond\_wait$)，在每次柜台服务完一个

消费者之后，$finish\_cus\_num++$，并发送一个signal，唤醒主程序中的忙等，当$finish_cus_num$等于消费者总数的时候，跳出while循环，程序结束。

3.2 消费者

每一个消费者之间是相互独立的，需要为每一个消费者创建一个进程。消费者的结构体定义如下：

Algorithm 2 struct Customer

```
$pthread\_t$ $cus\_pthread\_t$;//pthread var
$pthread\_mutex$ $_t$ $cus\_mutex\_t$;//mutex var
$pthread\_cond\_t$ $cus\_cond\_t$;//cond var
int $cus\_id$;//customer id
int $coun\_id$;//counter id
int $enter\_time$;//enter time
int $wait\_time$;//serve time
```

消费者的信息是从文件中读取进来的，因此为了实现模拟消费者在不同时刻进来的情形，需要在进程刚开始的时候sleep一段时间(各个消费者是不相同的)，在sleep一段时间后，按时间(执行)的顺序依次进入队列，以便后续过程。消费者在没有柜台服务的时候，是需要等到的，即用一个while循环，判断条件是该消费者的服务柜台号不为-1(初始值为-1，代表没有柜台服务)，当柜台和消费者匹配之后，由柜台修改消费者的柜台号，修改之后，消费者跳出循环，进入接受服务阶段，即sleep(服务)一段时间之后，发送一个signal给柜台，唤醒柜台，即表示服务已经结束。

3.3 程序流程

首先从$customer_data.dat$里面读取用户信息并保存在customer结构体数组中，然后进行初始化，为每一个柜台和每一个消费者进行初始化（创

建进程、锁变量初始化)，然后消费者进行入队列(queue)，开始上述两小节所描述的流程

Algorithm 3 bank counter-customer algorithm

```
algorithm describtion
    load data from ”$customer\_data.dat$”;
    initial varibales($pthread\_mutex_t, pthread\_cond_t$);
    create counter thread;
    create customer thread;
    for each counter
        record it’s id;
        if (num of customer $<=$ 0)
            stuck until num of customer $>$ 0;
        else
            arouse a customer;
            customer receive serve from that counter;
        wait and serve next customer;
    for each customer:
        come into bank in time order(after create, sleep for a while);
        wait until a counter arouse it;
        begin serve
        after serve, leave bank;
    when all customers are served, then program finfish;
```

4 运行结果及分析

4.1 生成测试样本

为了检验程序的正确性，我编写了一个生成随机样本的python程序，用来生成指定数目的顾客。(具体代码见附录)，下面展示随机生成的样本

(一部分):

```
3 1 1
1 1 5
2 2 5
3 2 8
4 3 5
5 5 10
6 8 5
7 8 9
8 9 6
9 9 1
10 10 3
11 11 2
12 14 6
13 15 9
```

5 程序运行结果

生成100个顾客，柜台数为5的运行结果:

```
customer id:0, enter at 1, served at 1, leave at 2,served by counter id:1
customer id:1, enter at 1, served at 1, leave at 6,served by counter id:0
customer id:2, enter at 2, served at 2, leave at 7,served by counter id:2
customer id:4, enter at 3, served at 3, leave at 8,served by counter id:4
customer id:3, enter at 2, served at 2, leave at 10,served by counter id:3
customer id:9, enter at 9, served at 10, leave at 11,served by counter id:3
customer id:6, enter at 8, served at 8, leave at 13,served by counter id:0
customer id:10, enter at 10, served at 11, leave at 14,served by counter id:3
customer id:5, enter at 5, served at 5, leave at 15,served by counter id:1
customer id:8, enter at 9, served at 9, leave at 15,served by counter id:4
customer id:11, enter at 11, served at 13, leave at 15,served by counter id:0
customer id:7, enter at 8, served at 8, leave at 17,served by counter id:2
customer id:14, enter at 15, served at 15, leave at 19,served by counter id:4
customer id:12, enter at 14, served at 14, leave at 20,served by counter id:3
customer id:13, enter at 15, served at 15, leave at 24,served by counter id:1
customer id:15, enter at 16, served at 17, leave at 26,served by counter id:0
customer id:20, enter at 21, served at 26, leave at 28,served by counter id:0
customer id:18, enter at 20, served at 21, leave at 28,served by counter id:4
customer id:16, enter at 19, served at 19, leave at 28,served by counter id:2
customer id:21, enter at 21, served at 28, leave at 30,served by counter id:0
customer id:17, enter at 20, served at 21, leave at 31,served by counter id:3
customer id:24, enter at 24, served at 30, leave at 32,served by counter id:0
customer id:23, enter at 22, served at 29, leave at 34,served by counter id:2
customer id:19, enter at 21, served at 24, leave at 34,served by counter id:1
customer id:25, enter at 28, served at 31, leave at 35,served by counter id:3
customer id:26, enter at 28, served at 32, leave at 35,served by counter id:0
customer id:22, enter at 22, served at 28, leave at 37,served by counter id:4
customer id:29, enter at 31, served at 35, leave at 37,served by counter id:3
customer id:30, enter at 32, served at 36, leave at 39,served by counter id:0
customer id:28, enter at 31, served at 35, leave at 39,served by counter id:1
customer id:27, enter at 29, served at 34, leave at 40,served by counter id:2
customer id:31, enter at 33, served at 37, leave at 43,served by counter id:4
customer id:36, enter at 34, served at 40, leave at 43,served by counter id:2
customer id:35, enter at 34, served at 39, leave at 44,served by counter id:0
customer id:33, enter at 34, served at 39, leave at 44,served by counter id:1
customer id:32, enter at 33, served at 37, leave at 45,served by counter id:3
customer id:34, enter at 34, served at 43, leave at 45,served by counter id:4
customer id:37, enter at 35, served at 43, leave at 47,served by counter id:2
customer id:40, enter at 39, served at 45, leave at 49,served by counter id:3
customer id:43, enter at 42, served at 50, leave at 52,served by counter id:3
customer id:38, enter at 35, served at 44, leave at 53,served by counter id:0
customer id:39, enter at 36, served at 44, leave at 54,served by counter id:1
customer id:42, enter at 40, served at 48, leave at 55,served by counter id:2
customer id:41, enter at 39, served at 46, leave at 55,served by counter id:4
customer id:48, enter at 46, served at 55, leave at 61,served by counter id:4
customer id:46, enter at 44, served at 54, leave at 61,served by counter id:1
customer id:44, enter at 43, served at 52, leave at 62,served by counter id:3
customer id:45, enter at 43, served at 53, leave at 63,served by counter id:0
customer id:47, enter at 46, served at 55, leave at 65,served by counter id:2
customer id:49, enter at 47, served at 61, leave at 65,served by counter id:4
customer id:51, enter at 50, served at 62, leave at 66,served by counter id:3
customer id:50, enter at 48, served at 61, leave at 66,served by counter id:1
customer id:54, enter at 53, served at 65, leave at 67,served by counter id:4
customer id:56, enter at 54, served at 68, leave at 71,served by counter id:4
customer id:57, enter at 54, served at 67, leave at 72,served by counter id:1
customer id:59, enter at 59, served at 72, leave at 73,served by counter id:1
customer id:58, enter at 56, served at 71, leave at 73,served by counter id:4
customer id:52, enter at 50, served at 64, leave at 74,served by counter id:0
customer id:53, enter at 51, served at 65, leave at 74,served by counter id:2
customer id:55, enter at 54, served at 66, leave at 74,served by counter id:3
customer id:63, enter at 64, served at 74, leave at 76,served by counter id:2
customer id:62, enter at 63, served at 74, leave at 79,served by counter id:0
customer id:60, enter at 62, served at 73, leave at 82,served by counter id:1
customer id:61, enter at 62, served at 73, leave at 82,served by counter id:4
customer id:66, enter at 65, served at 79, leave at 83,served by counter id:0
customer id:67, enter at 70, served at 82, leave at 84,served by counter id:1
customer id:65, enter at 65, served at 77, leave at 85,served by counter id:2
customer id:64, enter at 64, served at 75, leave at 85,served by counter id:3
customer id:68, enter at 70, served at 83, leave at 86,served by counter id:4
customer id:70, enter at 71, served at 85, leave at 88,served by counter id:1
customer id:72, enter at 72, served at 85, leave at 90,served by counter id:3
customer id:74, enter at 72, served at 88, leave at 91,served by counter id:1
customer id:77, enter at 74, served at 91, leave at 93,served by counter id:1
customer id:69, enter at 71, served at 83, leave at 93,served by counter id:0
customer id:75, enter at 73, served at 90, leave at 94,served by counter id:3
customer id:71, enter at 71, served at 85, leave at 95,served by counter id:2
customer id:76, enter at 74, served at 93, leave at 95,served by counter id:1
customer id:73, enter at 72, served at 86, leave at 96,served by counter id:4
customer id:79, enter at 76, served at 94, leave at 98,served by counter id:3
customer id:81, enter at 80, served at 96, leave at 101,served by counter id:1
customer id:84, enter at 81, served at 101, leave at 102,served by counter id:1
customer id:82, enter at 80, served at 96, leave at 102,served by counter id:4
customer id:78, enter at 76, served at 93, leave at 102,served by counter id:0
customer id:86, enter at 82, served at 102, leave at 104,served by counter id:4
customer id:80, enter at 79, served at 95, leave at 105,served by counter id:2
customer id:87, enter at 83, served at 103, leave at 108,served by counter id:0
customer id:83, enter at 81, served at 99, leave at 108,served by counter id:3
customer id:85, enter at 82, served at 102, leave at 108,served by counter id:1
customer id:89, enter at 84, served at 105, leave at 109,served by counter id:2
customer id:88, enter at 83, served at 104, leave at 111,served by counter id:4
customer id:96, enter at 91, served at 112, leave at 113,served by counter
```

```
id:4
customer id:90, enter at 86, served at 108, leave at 115,served by counter id:0
customer id:91, enter at 88, served at 108, leave at 115,served by counter id:3
customer id:93, enter at 89, served at 108, leave at 115,served by counter id:1
customer id:92, enter at 89, served at 110, leave at 117,served by counter id:2
customer id:95, enter at 91, served at 115, leave at 120,served by counter id:0
customer id:94, enter at 91, served at 113, leave at 121,served by counter id:4
customer id:99, enter at 95, served at 117, leave at 122,served by counter id:2
customer id:97, enter at 92, served at 115, leave at 122,served by counter id:3
customer id:98, enter at 92, served at 115, leave at 124,served by counter id:1
```

结果分析:

从上面的运行结果来看，比如对于柜台1，先服务顾客0并于时刻2结束服务，然后在时刻5服务顾客5并于时刻15结束，再在时刻15服务顾客13并于时刻24结束，依次类推，对每一个柜台都是这样的，运行结果没问题。

6　思考题

6.1　柜员人数和顾客人数对结果分别有什么影响

顾客人数很多，而柜台人数很少的时候，最后一名顾客结束服务的时刻，和柜台数量大致程反比。当柜台数较少，同一时间能服务的的顾客数

就越少，吞吐量越小，顾客等待的时间也就越长。

当柜台数较多时，同一时间能服务的顾客就越多，系统的吞吐量就越大，顾客等待时间就越短。

6.2 实现互斥的方法有哪些？各自有什么特点？效率如何？

实现互斥的方法有：禁止中断、自旋锁、互斥锁、信号量

禁止中断：在内核态小代码段使用，用户态没有这样的接口，效率高；

自旋锁：忙等待，特点是死循环占满CPU，效率低；

互斥锁：用阻塞，不会浪费CPU资源，效率高；

信号量：可以实现多个生产、消费的互斥，有P、V两种操作，效率高；

7 实验总结

为了完成这次实验，我翻看了一些linux多线程编程的参考书，查阅了很多资料，这让我对linux更加熟悉，同时对线程进程的理解更加深刻。

虽然这次实验的难度并不是很大，但它带给了我全新的体验，因为之前从来没进行过多线程的编程，调试起来觉得特别困难，所以这对我来说是一个比较大的挑战，后来我分段输出，根据输出结果来分析，最终成功完成实验。

8 其他

8.1 运行代码指令

1. gcc -o test test.c -lpthread
2. ./test

8.2 附录

附录1：test.c（算法代码）

附录2：*gen_data.py*（生成用户数据代码）

附录1：主程序(test.c)

test.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>
#include <time.h>

/**********************************************/
//predefine
#define COUNTER_NUM 5
#define CUSTOMER_NUM_MAX 500
/**********************************************/


/**********************************************/
//struct define
struct Counter
{
        pthread_t c_pthread_t;//pthread var
        pthread_mutex_t c_mutex_t;//mutex var
        pthread_cond_t c_cond_t;//cond var
};

struct Customer
{
        pthread_t cus_pthread_t;//pthread var
        pthread_mutex_t cus_mutex_t;//mutex var
        pthread_cond_t cus_cond_t;//cond var
        int cus_id;//customer id
        int coun_id;//counter id
        int enter_time;//enter time
        int wait_time;//serve time
};

struct queue_list
{
        int cus_id;
};

/*********************************************/

/*********************************************/
//predefine params
pthread_mutex_t que_lock;
pthread_mutex_t finish_cus_mutex_t;
pthread_mutex_t queue_t;
pthread_cond_t finish_cus_cond_t;
struct Counter counter[COUNTER_NUM];
struct Customer customer[CUSTOMER_NUM_MAX];
struct queue_list queue[CUSTOMER_NUM_MAX];
sem_t customer_wait;
int CUSTOMER_NUM = 0;
int finish_cus_num = 0;
int q_last = 0;
int q_first = 0;
const char* filename = ”customer_data.dat”;
```

```
struct timeval c_begin,c_end;
/*******************************************/

/*******************************************/
//function predefine
void init_counter();
void init_customer_();
void readf(const char*);
int queue_push(int);
void* counter_server(void*);
void* customer_server(void*);
struct queue_list* queue_pop();
double difftime(clock_t,clock_t);
/*******************************************/

int main()
{
        pthread_mutex_init(&que_lock,NULL);//create mutex for queue
        sem_init(&customer_wait,0,0);//create sem signal
        readf(filename);//read customer data from filename,and save in customer
        gettimeofday(&c_begin,NULL);
        init_counter();//init counter
        init_customer_();//init customer

        //init fi-mutex, queue-mutex and fi-cond
        pthread_mutex_init(&finish_cus_mutex_t,NULL);
        pthread_mutex_init(&queue_t,NULL);
        pthread_cond_init(&finish_cus_cond_t,NULL);

        pthread_mutex_lock(&finish_cus_mutex_t);//lock fi-mutex
        //printf("fin_mutex is locked\n");
        while(finish_cus_num != CUSTOMER_NUM)
        {
                //waiting a signal to continue the while
                //the signal must be released by counter(when a customer finished the server)
                pthread_cond_wait(&finish_cus_cond_t,&finish_cus_mutex_t);
        }
        pthread_mutex_unlock(&finish_cus_mutex_t);//unlock fi-mutex
        //printf("fin_mutex is unlocked\n");
        return 0;
}

/*********************************************/
//counter function
void* counter_server(void* id)
{
        int counter_id = id;
        int serve_time = 0;//every counter has a serve time,to record current time
```

```
    //printf("counter id:%d\n",counter_id);
    int start_time = 0;
    int end_time = 0;
    struct timeval start,end;
    int flag = 1;
    while(1)
    {
        if(flag == 1)
        {
            gettimeofday(&start,NULL);
            //printf("counter id:%d begin work at %d\n",counter_id,start);
            flag = 0;
        }
        sem_wait(&customer_wait);//if sem>0, sem-- and continue, if sem <=0, sutck in here until sem_post make sem > 0
        struct queue_list* q_next = queue_pop();//read first member in the queue
        //printf("counter_id:%d,qfirst:%d,qlast:%d\n",id,q_first,q_last);
        if(q_next == NULL)//if queue is NULL, continue waiting
        {
            //printf("no customer\n");
            continue;
        }
```

```
                if(customer[q_next->cus_id].coun_id != -1)//if customer has already served by a counter
                {
                        //printf("customer id:%d  already have a counter\n", q_next->cus_id);
                        continue;
                }
                //printf("customer id:%d in serve\n", q_next->cus_id);
                //now,first customer in the queue receive the serve by current counter
                pthread_mutex_lock(&customer[q_next->cus_id].cus_mutex_t);//lock customer, prevent it is served by other counter
                int current_cus_id = q_next->cus_id;//save the customer id
                customer[current_cus_id].coun_id = counter_id;//adjust customer's coun_id
                pthread_mutex_lock(&counter[counter_id].c_mutex_t);//lock counter, prevent it serve other customer
                //printf("counter id:%d lock customer id: %d\n",counter_id, current_cus_id);
                pthread_cond_signal(&customer[
current_cus_id].cus_cond_t);//signal that customer, let it in serve
                pthread_mutex_unlock(&customer[current_cus_id].cus_mutex_t);//unlock customer
                //printf("counter id:%d unlock customer id: %d\n",counter_id,current_cus_id);
                pthread_cond_wait(&counter[counter_id].c_cond_t,&counter[counter_id].c_mutex_t);//if customer finish it serve, send a signal to counter
                //printf("customer id:%d server over\n",current_cus_id);
                gettimeofday(&end,NULL);
                pthread_mutex_unlock(&counter[counter_id].c_mutex_t);//customer serve finished, unlock counter
                end_time = (end.tv_usec-start.tv_usec)/1000;
                //printf("counter id: %d,customer id:%d end time:%d\n",id,current_cus_id,end_time);
                start_time = end_time - customer[current_cus_id].wait_time;//compute finished serve time
                //time = enter_t + customer[current_cus_id].wait_time;
                printf("customer id:%d, enter at %d, served at %d, leave at %d, served by counter id:%d\n",(current_cus_id),customer[current_cus_id].enter_time,start_time,end_time,counter_id);

                //after serving, judging if there is no customer waiting in queue
                pthread_mutex_lock(&finish_cus_mutex_t);
                finish_cus_num++;//aftering a customer served,
                pthread_cond_signal(&finish_cus_cond_t);
                pthread_mutex_unlock(&finish_cus_mutex_t);
                pthread_mutex_unlock(&counter[counter_id].c_mutex_t);//customer serve finished, unlock counter

        }
}
/***********************************************/

/***********************************************/
//customer function
void* customer_server(void* customer_data)
{
        struct Customer* cus = (struct Customer*) customer_data;
        //printf("customer id:%d,enter time:%d\n",cus->cus_id,cus->enter_time);
    usleep(cus->enter_time*1000);
    //printf("customer id:%d,enter time:%d\n",cus->cus_id,cus->enter_time);

    pthread_cond_init(&customer[cus->cus_id].cus_cond_t,NULL);
    pthread_mutex_init(&customer[cus->cus_id].cus_mutex_t,NULL);

    pthread_cond_t* cus_cond_p = &customer[cus->cus_id].cus_cond_t;
    pthread_mutex_t* cus_mutex_p = &customer[cus->cus_id].cus_mutex_t;

    pthread_mutex_lock(cus_mutex_p);// lock customer
    int num = queue_push(cus->cus_id);
    sem_post(&customer_wait);
    //printf("customer id:%d is attempting arosing a counter\n",cus->cus_id);
    while(customer[cus->cus_id].coun_id == -1)//wait until a counter is unbusy
    {
        //printf("customer id:%d is waiting\n",cus->cus_id);
        pthread_cond_wait(cus_cond_p,cus_mutex_p);//wait a counter to arouse it
    }
    //printf("customer id:%d is aroused by counter id:%d\n",cus->cus_id,customer[cus->cus_id].coun_id);
    //a counter respond
    int counter_id = customer[cus->cus_id].coun_id;
    //printf("customer id:%d is svered by counter id:%
d\n",cus->cus_id,counter_id);
    //printf("counter id %d serve customer id %d\n",counter_id,cus->cus_id);
    pthread_mutex_lock(&counter[counter_id].c_mutex_t);//lock counter
    usleep(customer[cus->cus_id].wait_time*1000);//stimulate serve
    pthread_cond_signal(&counter[counter_id].c_cond_t);

    pthread_mutex_unlock(&counter[counter_id].c_mutex_t);//unlock counter
    //printf("customer id:%d servering over\n",cus->cus_id);


    pthread_mutex_unlock(&(*cus_mutex_p));//unlock customer
}
/********************************************/


/********************************************/
//queue funtion
struct queue_list* queue_pop()
{
        pthread_mutex_lock(&queue_t);
        if(q_first == q_last)
        {
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣pthread_mutex_unlock(&queue_t);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣return␣NULL;
␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣else
␣␣␣␣␣␣␣␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣q_first++;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//printf("q_first:%d\n",q_first);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣pthread_mutex_unlock(&queue_t);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣return␣&queue[q_first-1];
␣␣␣␣␣␣␣␣}
}

int␣queue_push(int␣id)
{
␣␣␣␣␣␣␣␣pthread_mutex_lock(&queue_t);
␣␣␣␣␣␣␣␣queue[q_last].cus_id=id;
␣␣␣␣␣␣␣␣q_last++;
␣␣␣␣␣␣␣␣//printf("q_last:%d\n",q_last);
␣␣␣␣␣␣␣␣pthread_mutex_unlock(&queue_t);
␣␣␣␣␣␣␣␣return␣(q_last␣-␣1);
}
/**********************************************/



/**********************************************/
//initial␣function
void␣readf(const␣char*␣filename)
{
FILE* f = fopen(filename,"rb");
int cus_id, enter_time, wait_time;

while(fscanf(f,"%d %d %d\n",&cus_id,&enter_time,&wait_time) != EOF)
{
//printf("CUSTOMER_NUM:%d\n",CUSTOMER_NUM);
customer[CUSTOMER_NUM].cus_id = cus_id;
customer[CUSTOMER_NUM].enter_time = enter_time;
customer[CUSTOMER_NUM].wait_time = wait_time;
customer[CUSTOMER_NUM].coun_id = -1;
printf("CUSTOMER_NUM:%d,enter_time%d,wait time%d,coun id: %d\n",cus_id,enter_time,wait_time,customer[CUSTOMER_NUM].coun_id);
CUSTOMER_NUM++;

}
printf("load data successfully\n");
}

void init_customer_()
{
int flag_error = 0;
for(int i = 0; i < CUSTOMER_NUM; i++)
{
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣flag_error␣=␣pthread_create(&customer[i].cus_pthread_t,NULL,customer_server,(void*)&customer[i]);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣flag_error␣=␣pthread_mutex_init(&customer[i].cus_mutex_t,NULL);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣flag_error␣=␣pthread_cond_init(&customer[i].cus_cond_t,NULL);
␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣if(flag_error␣==␣0)
␣␣␣␣␣␣␣␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//printf(”successfully␣init␣counter\n”);
␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣else
␣␣␣␣␣␣␣␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣printf(”init␣customer␣with␣error\n”);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣exit(−1);
␣␣␣␣␣␣␣␣}
}

void␣init_counter()//initial␣counter
{
␣␣␣␣␣␣␣␣int␣flag_error␣=␣0;
␣␣␣␣␣␣␣␣for(int␣i␣=␣0;i␣<␣COUNTER_NUM;i++)
␣␣␣␣␣␣␣␣{

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣flag_error␣=␣pthread_mutex_init(&counter[i].c_mutex_t,NULL);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣flag_error␣=␣pthread_cond_init(&
counter[i].c_cond_t,NULL);
                flag_error = pthread_create(&counter[i].c_pthread_t,NULL,counter_server,(void*)i);
        }
        if(flag_error == 0)
                {
                        //printf("successfully init counter\n");
                }
        else
        {
                printf("init counter with error\n");
                exit(-1);
        }
}
/**********************************************/
```

附录2：生成用户数据程序(datagen.py)

datagen.py

```python
import random
file = "customer_data.dat"

customer_num = 100

data = []
k = 1
for i in range(customer_num):
    s = []
    s.append(random.randint(1,100))
    s.append(random.randint(1,10))
    data.append(s)

data.sort(key = lambda x:x[0])
#print data

with open(file,'w') as f:
    for i in range(customer_num):
        f.write(str(i));
        f.write(" ")
        f.write(str(data[i][0]))
        f.write(" ")
        f.write(str(data[i][1]))
        f.write('\n')
```