# 目录

# 实验目的

1. 了解并正确使用 Linux 的 IPC 机制
2. 利用 Unix/Linux 的 IPC 机制仿真一个音乐厅门票订售系统

# 实验要求

1. 系统要包含若干个售订票代理，这些代理连接同一台计算机，每个代理相当于一个进程
2. 系统的设计要考虑到系统的安全以及容错能力，例如：
   a) 代理企图订/售已经占用的座位，系统会禁止，并且输出“座位已被占用”的提示信息
   b) 如果对不存在的座位进行操作，系统输出出错信息
3. 每个代理的每次操作都要用一定的延时，延时的配置在系统输入中确定，系统输入格式如下

```
n     //  音乐厅座位排数
m     //  每排座位数
k     //  代理个数
rvt   //  预约的有效期（单位分钟）
agent 1
reserve reserve_time (单位秒)
ticket ticket_time   (单位秒)
cancel cancel_time   (单位秒)
check_customer check_time (单位秒)
  :
有效的交易列表
  :
```

4. 每个代理处理的任务格式如下:

```
reserve <座位描述> name_of_customer  // 为某客户预订指定座位的票
ticket <座位描述> name_of_customer   // 为某客户购买指定座位的票
cancel <座位描述> name_of_customer   // 为某客户取消指定座位的预订
show all name_of_customer            // 显示某客户的预订或购票情况
```

5. 客户的座位选择是多样的，要求包含以下座位选择格式

```
A 2 // 第A行，第2座
A 4 7 // 第A行，第4到7座
A // 第A行
A 2r // 从第A行开始的2行
A 2c // 第A行，2个座位（不在意具体列）
6 // 6个座位（不在意具体行和列）
```

6. 预约的座位只保留一定的时间，超出时间将会自动取消
7. 购票之前无需预约
8. 取消操作只对预定但未被售出的座位有效
9. 客户名字不能包含空格，且首字母不能是数字
10. 每个交易完成后都要给出交易的信息摘要，并且再系统结束后显示座位订售情况

# 实验使用的 IPC 及其封装

Linux 的 IPC 机制也就是多进程通信的意思，是多进程通信的一种方法。在 Linux 中有多种进程间通信的方法：半双工管道、命名管道、消息队列、信号、信号量、共享内存、内存映射文件，套接字等等。本次实验主要用到课程中学到的：

1. fork，wait
2. 共享内存
3. 信号灯

**进程创建与撤销**

`pid_t fork()`

创建一个内容和自身相同的子进程，两个进程的用户空间独立，平等调度，如果调用成功，将会返回 0

`pid_t wait(int *__stat_loc)`

等待进程的一个子进程结束，如果有进程结束，将返回其进程 id，如果操作失败，返回-1

**共享内存**

共享内存是多个进程之间共享内存区域的一种方法，使用 IPC 对进程创建一个特殊的地址范围，其他进程通过地址映射的方法，共同访问一段内存空间，而这部分内存在各个进程中的地址可以不一样，如果一个进程对这部分内存改动，其他进程会接受到这些改动。

共享内存是 IPC 中最快捷的方式，因为共享内存没有中间的通信过程，而管道、消息队列则是需要消息的机制对内容进行转换。


在本实验中，对于音乐厅的座位，使用共享内存的形式存储。

共享内存主要涉及以下操作。

```
int shmget(key_t __key, size_t __size, int __shmflg)
```

创建共享内存，需要输入共享内存的一个唯一标识，共享内存大小，以及创建共享内存时的存取权限

```
void *shmat(int __shmid, const void *__shmaddr, int __shmflg)
```

根据共享内存的标识，获取共享内存的地址映射，可以手动设置地址的格式，也可以通过系统自动分配地址映射(将 shmaddr 设为 NULL)，允许设置这段内存空间的读写权限

```
int shmdt(const void *__shmaddr)
```

切断地址映射，当前进程如果调用这个函数，将不能再访问共享内存区，但是不会销毁共享内存区

**信号灯**

信号灯是一种计数器，由计算机前辈 dijstra 发明，用于控制多进程对临界资源的访问，再本次项目中，音乐厅的座位是临界资源，使用 linux 中提供的信号灯工具可以设计互斥锁，防止多个进程同同一个座位同时操作。信号灯主要有 p 操作(自增)和 v 操作(自减)

Linux 中提供以下信号灯工具

```
int semget(key_t __key, int __nsems, int __semflg)
```

获取信号灯，可以一次创建多个信号灯

```
int semop(int __semid, sembuf *__sops, size_t __nsops)
```

对指定信号灯集合的对应的信号灯进行操作，操作和信号灯指定通过__sops 确定。sembuf 是一个内置的结构体，通过这个结构体指定需要操作的信号灯和需要的操作

```
struct sembuf
{
  unsigned short int sem_num; /* semaphore number */
  short int sem_op;     /* semaphore operation */
  short int sem_flg;    /* operation flag */
};
```

通过 semop 的设计，可以实现信号灯的 pv 操作，我的实现方法如下

```
void sem_wait(int sid, int sn)//p 操作
{
    struct sembuf op;
    op.sem_num=sn;
```

```
    op.sem_op=-1;
    op.sem_flg=0;
    if(semop(sid, &op, 1)==-1){
        sem_exception.set("sem_wait fail");
        throw sem_exception;
    }
}

void sem_signal(int sid, int sn) //v 操作
{
    struct sembuf op;
    op.sem_num = sn;
    op.sem_op = 1;
    op.sem_flg = 0;
    if (semop(sid, &op, 1) == -1)
    {
        sem_exception.set("sem_signal fail");
        throw sem_exception;
    }
}
```

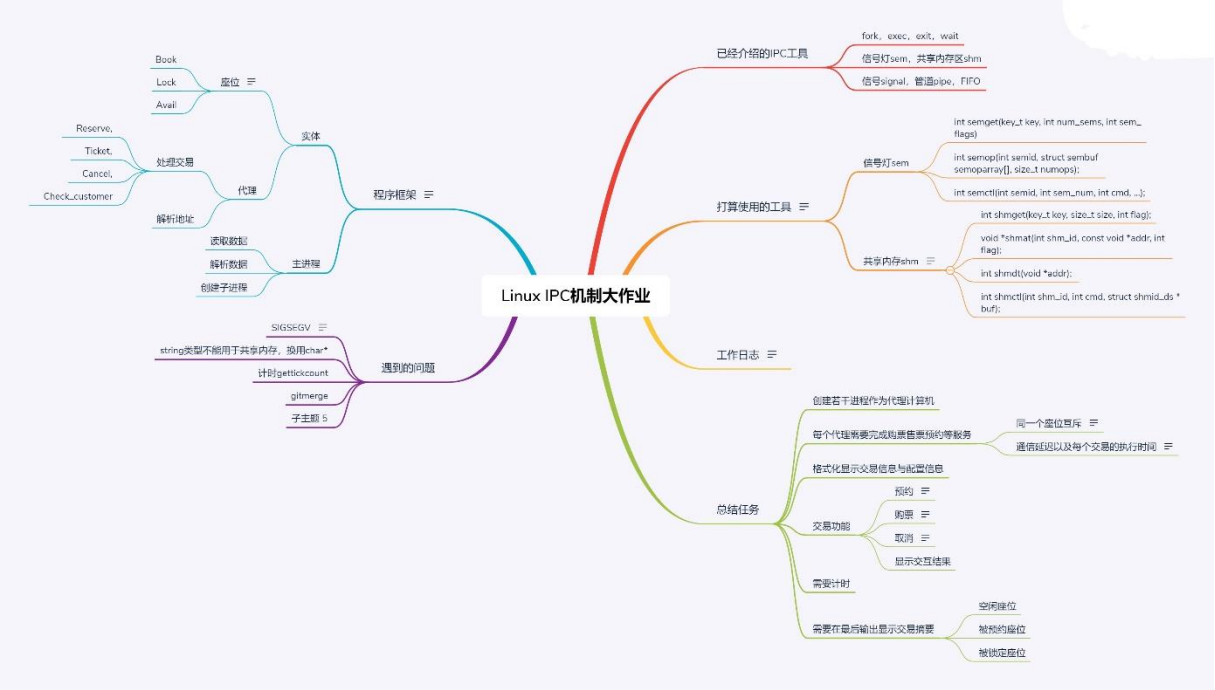
```
/* 创建并初始化信号量 */
int create_sem(const char * path, int id, int sem_num, int val=1); // 多个信号量
/* 信号量p操作*/
void sem_wait(int sid, int sn);

/* 信号量v操作 */
void sem_signal(int sid,int sn);

/* 删除信号量 */
void remove_sem(int sid);

/* 创建共享内存区 */
int create_shm(const char *path, int id, size_t shmsize);

/* 删除共享内存区 */
void remove_shm(int shm_id);

/* 获取共享内存区地址 */
void * connect_shm(int shm_id);

/* 删除共享内存映射 */
void * cut_shm(const void * addr);

}
```

ipc_util.h

考虑到实现的便捷性和共享内存与信号量的执行效率，我认为在本实验中使用共享内存与信号量实现是合适的。ipc 机制实现代码在我的 ipc_util.cpp 中可以找到

# 实现方法

本次项目的实验流程，以及我对实验要求的分析可以用下图概括

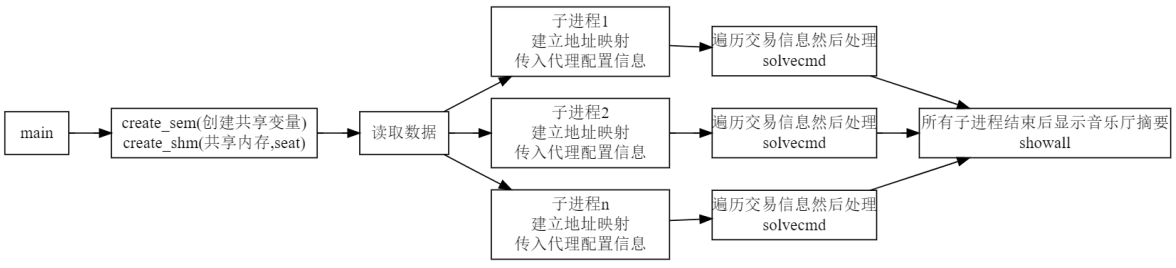



# 实验环境

1. Linux 版本：ubuntu18.04
2. C++版本：C++11

# 具体实现

### 执行流程



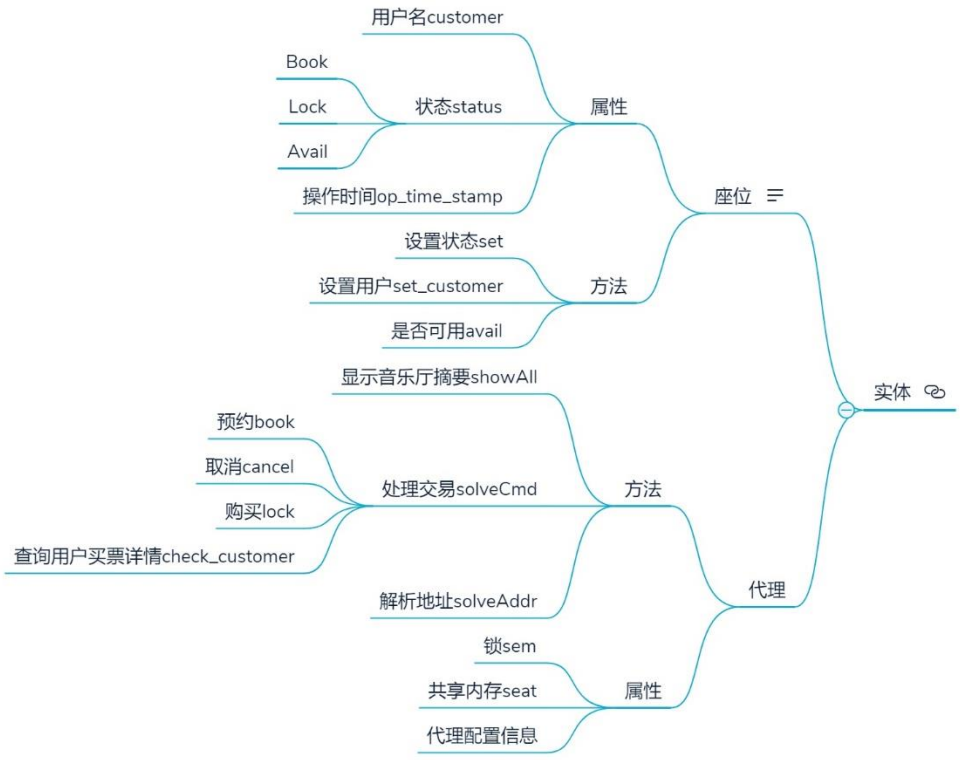

程序的执行入口是 main，按照进程扇的形式创建子进程，每次读取完一个代理的配置信息后，会得到一个交易信息的列表，然后会**创建子进程**，此时子进程会复制父进程已经读取号的交易信息列表，子进程继续处理这个列表，父进程**清空交易信息列表**，继续读取数据，再次读取完一个代理的信息后又创建一个子进程，**重复上述操作**，直到读取完所有数据，父进程**进入等待**状态，等所有的子进程结束后，**输出音乐厅的交易摘要**，父进程子进程的工作总结如下：

1. 父进程：

a) 读取系统输入，转换为 Transction(交易结构体)数组
b) 读取完一个 agent 的输入后，调用 fork 创建子进程
c) 清空 Transction 数组，继续读取输入
d) 读取完后调用 wait()，等待所有子进程结束后销毁子进程
e) 子进程结束后调用 showall()函数，输出摘要信息

2. 子进程：
   a) 读取复制父进程过来的 Transction 数组，遍历每一条交易
   b) 调用 solvecmd()处理交易
   c) 遍历完交易后调用 exit 退出
3. 子进程和父进程构成了一个进程扇，这种结构有利于进程的管理，子进程最后必定会被销毁，不会产生孤儿进程。

**实体**




**Seat 实体**提供了座位的信息意义对应的操作，需要注意的是，Seat 实体的方法并不保证线程安全，只是对 Seat 对象进行操作。

```
class Seat{
    time_t op_time_stamp;
    Stat status;
    char customer[USER_LEN];
public:
    Seat(){…
    }
/* 设置座位状态 */
    void set(Stat stat){…
    }
/* 设置座位所属用户 */
    void set_customer(string customer){…
    }
/*是否空闲*/
    bool avail(double limit, string &customer);
/*获取最后一次操作时间*/
    time_t get_time_stamp();
/*获取购票者信息*/
    string get_customer();
/* 获取状态*/
    Stat get_stat();
};
```

Seat 实体

每个座位都有自身的状态，对应的用户名(如果没有，置空)，最后一次预约时间

其中一个比较特殊的函数时 avail，这个函数会在每次调用时更新座位状态，如果座位已经超过了预约的时限，则置为空闲状态，然后再检查座位是否对当前用户可用，可用则放回 true，否则为 false

**Concert 实体**提供了代理需要的一切操作，该音乐厅的主要处理逻辑都在这里。

```
class Concert{
    int agent_id;
    // 需要m*n+1个锁，第0个作为全局锁
    key_t sid;
    int n;
    int m;
    double rvt;
    Seat * seats; //一维替二维
    bool enable;
    time_t start_tick;
    // 处理延迟时间
    double reserve_time;double  ticket_time;double cancel_time;double check_time;
public:
    /* 需要传入音乐厅配置信息*/
    Concert(int n, int m, double rvt);
    /* 需要传入代理配置信息，绑定锁和座位(共享内存) */
    void initConcert(int agent,int sid, Seat *seats,double reserve_time,double  ticket_time,double cancel_time,double check_time);
    /* 以下函数返回操作是否有效 处理单个座位*/
    bool book(int i, int j, string &customer,Stat &stat);
    bool cancel(int i, int j, string &customer,Stat &stat);
    bool lock(int i, int j, string &customer,Stat &stat);
    bool check_customer(int i,int j,string &customer,Stat &stat);

/* 解析地址，如果地址无效则返回false，地址有效则返回true…
*/
    AddrType solveAddr(string seats, vector<Pos> &result_pos);

    /* 对应一条命令的处理,并且把结果输出 */
    /* 输出一条命令的提示信息，该信息将被专门用于显示的进程读取 */
    bool solveCmd(Transaction tx);

    SeatInfo semGetSeatStat(int i, int j, double rvt);
    void showAll();
```

**初始化函数**

void initConcert(int agent,int sid, Seat *seats…)

传入代理的配置信息，以及共享内存中音乐厅座位的地址映射，并为每一个座位分配一个信号量

**处理逻辑函数**

bool book(int i, int j, string &customer,Stat &stat);

bool cancel(int i, int j, string &customer,Stat &stat);

bool lock(int i, int j, string &customer,Stat &stat);

bool check_customer(int i,int j,string &customer,Stat &stat);

这四个函数在处理座位的操作流程基本一样，都会先更新座位状态，更新状态使用的是座位对象的 avil 函数更新，返回操作成功与否，并把座位操作之前的状态存入 stat，这四个函数是线程安全的，**每个座位有一个互斥锁，内部使用了互斥地访问每一个座位**

```
bool Concert::book(int i , int j, string & customer,Stat &stat){
    assert(enable);
    if(i >= n || j >= m || i <0 || j < 0) return false;
    bool result = false;
    sem_wait(sid,i*m+j);
    // 更新
    (seats+i*m+j)->avail(rvt,customer);
    stat = (seats+i*m+j)->get_stat();
    if(stat != Book && stat != Lock){
        (seats+i*m+j)->set(Book);
        (seats+i*m+j)->set_customer(customer);
        result = true;
    }
    sem_signal(sid,i*m+j);
    return result;
}
```

book 函数内部，先更新座位信息，然后判断是否可预约

**地址处理函数**

AddrType solveAddr(string seats, vector<Pos> &result_pos);

这个函数会对预设的几种座位输入格式进行解析，解析出座位类型后返回给 solvecmd 函数，如果在地址解析过程中可以得到所有需要地座位，则会把座位信息存入 result_pos，该函数具体实现在后续**重点函数**章节讲解

**业务逻辑函数**

bool solveCmd(Transaction tx);

这个函数是代理最重要的一个接口，只要传入交易信息即可自动调用内部函数完成操作，并输出显示，Transcation 是一个特殊的结构体，保存了交易所需的所有信息，如交易类型，座位选择，用户名等。该函数操作逻辑比较长，具体逻辑详见**重点函数**章节

void showAll();

这个函数用于输出音乐厅摘要

**重点函数**

## 实现不同地址类型解析功能

AddrType solveAddr(string seats, vector<Pos> &result_pos);
该函数可以识别并解析 6 中地址格式输入：

1. 一行，如：A
2. 特定的一个座位，如：A 2
3. 一行中的特定几个座位，如：A 4 7，表示(A,4~7)
4. 一行中的随机几个位置，如：A 2c，表示在 A 行找两个位置
5. 一行中的一个位置，如：A 2，表示(A,2)
6. 任意几个位置，如：6，表示找 6 个座位

```
/* 几种地址解析的正则表达式 */
regex oneRow("[A-Z] ");
regex oneRowOnePos("[A-Z] \\d+ ");
regex oneRowServerlPos("[A-Z] \\d+ \\d+ ");
regex oneRowRandPos("[A-Z] \\d+c ");
regex serverlRow("[A-Z] \\d+r ");
regex serverlPos("\\d+ ");
```

另外设置返回地址的类型，供 solvecmd 函数判断

```
enum AddrType{
    OneRow,
    OneRowOnePos,
    OneRowServerlPos,
    OneRowRandPos,
    ServerlRow,
    ServerlPos,
    Error
};
```

如果可以在 solveAddr 函数中获取座位的话，那么 result_pos 就是该结果，否则，result_pos 只记录辅助 solvecmd 设置座位的信息。

OneRow, 返回(k,0)
OneRowOnePos,返回(i,j)
OneRowServerlPos,返回一行中的所有连续的几个座位
OneRowRandPos,(k,m)k 行 m 个
ServerlRow,返回(k,m)，k 行以后 m 列 m 只能正数
ServerlPos,返回(k,0)

## 实现交易处理功能

bool Concert::solveCmd(Transaction tx)

该函数的执行流程如下：

1. 通过 tx.tx 确定交易类型，如果交易类型不合法，则会返回 false
2. 调用 solveAddr(tx.pos)解析地址类型，根据地址类型确定如何遍历座位
3. 根据不同的交易类型进行对应的处理，并使用一个字符 IO msg 收集输出信息：
   a) Reserve:调用 book 函数，操作成功则输出对应信息到 msg，否则将错误信息输出到 msg
   b) Ticket:调用 buy 函数，操作成功则输出对应信息到 msg，否则将错误信息输出到 msg
   c) Cancel:调用 cancel 函数，操作成功则输出对应信息到 msg，否则将错误信息输出到 msg
   d) show:遍历所有的座位，获取所有座位的信息，收集到 msg 中
4. 将 msg 的字符串送入 sendmsg 函数中，该函数可以互斥地调用 printf，输出字符到终端
5. 根据交易类型做相应的延时

```
bool Concert::solveCmd(Transaction tx){
    assert(enable);
    vector<Pos> result;
    ostringstream msg;
    msg << "---------------" << tx.customer <<" " << TxArr[tx.tx] << "-------------------\n";
    msg << "request position: " << tx.pos << '\n';
    msg << "---------------result---------------\n";
    string fail_msg = "Error:\n";
    switch (tx.tx)
    {…
    }
    if(tx.tx == Show){…
    }


    else{…
    }

    // 睡眠
    double sleep_time = 0;
    switch (tx.tx)
    {
    case Reserve:…
    case Cancel:…
    case Ticket:…
    case Show:…
    default:…
    }

    msg << "agent " << agent_id << " sleep for " << sleep_time << " seconds\n";
    sendMsg(msg.str());

    msg.clear();
    sleep(sleep_time);
    return true;
```

solvecmd 部分内容

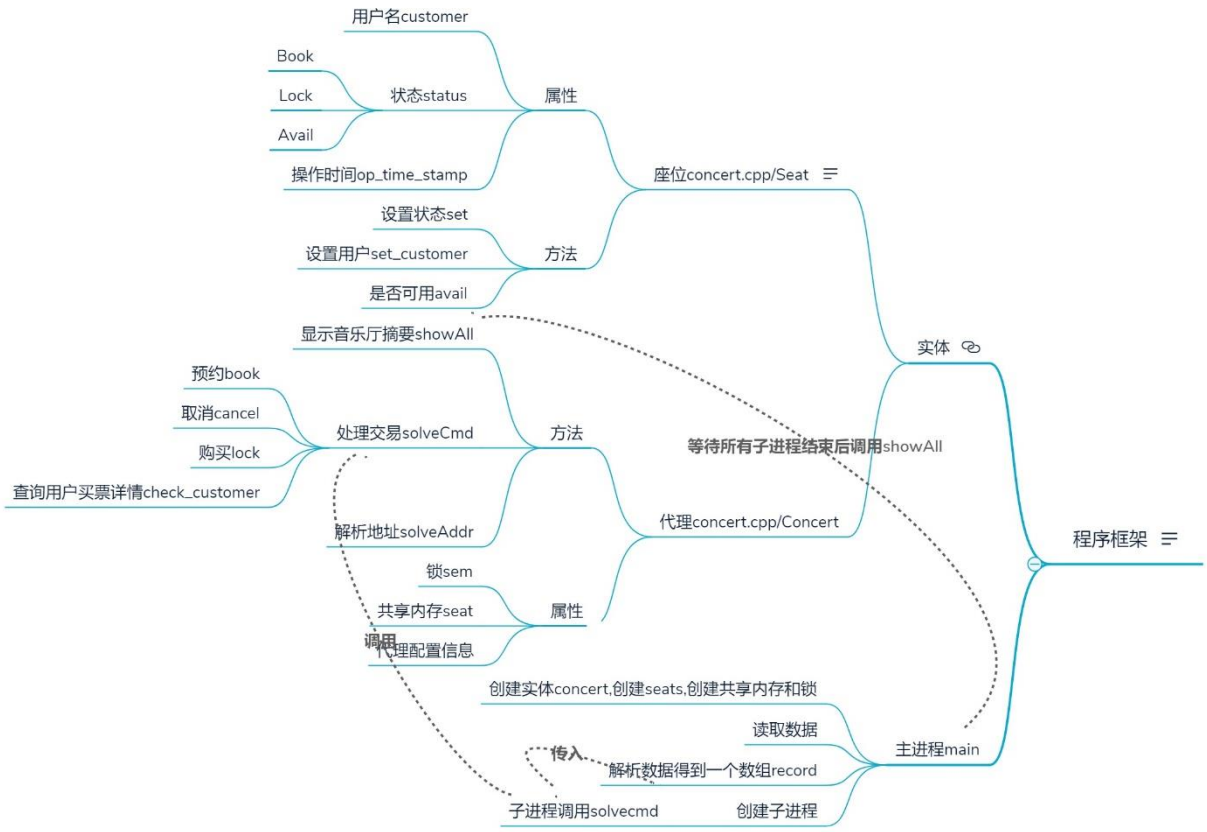
## 实现互斥输出到终端功能

由于不确定 printf 是否是线程安全的，所以为了保证安全，在输出摘要信息时，使用 sendmsg 函数，该函数内部使用互斥锁保证 printf 被互斥访问

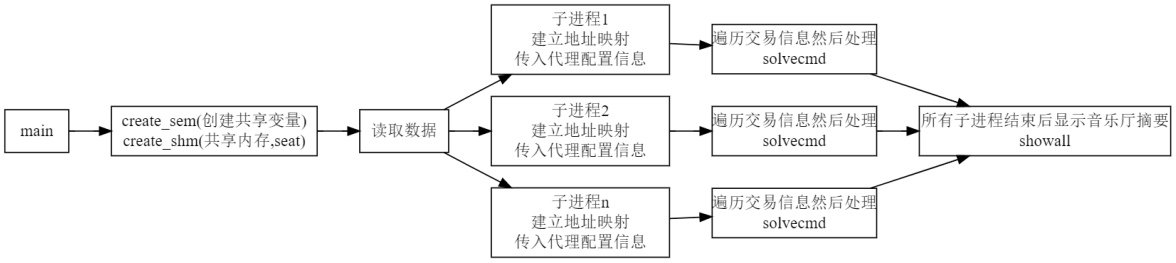
```
// 处理消息后回显，使用write函数在linux内核版本3.14后都是原子操作
void Concert::sendMsg(const string msg){
    // 传入多一把锁作为互斥锁
    sem_wait(sid,m*n);

    printf("================agent: %d Tick: %ld==============\n",this->agent_id,long(difftime(time(NULL),this->start_tick)));
    printf("%s",msg.c_str());

    sem_signal(sid,m*n);
    printf("====================good bye=======================\n\n");

}
```

# 程序架构

整个程序框架和执行流程可以总结如下：




程序框架，注意,每个座位分配一把互斥锁,包装操作的互斥性




执行流程，进程扇结构

# 测试结果

## 不同地址的解析功能测试

测试文件是 test/difference_type_pos.txt
内容如下：

| 代理 | agent1 | agent2 |
|---|---|---|
| 交易内容，<br>音乐厅 10 排 10 列 | ticket "A 2c" Barry | ticket "D" Barry |
| | ticket "B 2" Ellen | ticket "E 2r" Barry |
| | ticket "C 2 5" Barry | ticket "6" Barry |
| | ticket "F 11" Barry<br>（无效地址） | Ticket "K 5" Barry<br>（无效地址） |

**测试结果**：检查后发现，各个座位类型均被解析正确，以下是测试的输出(详细输出结果见附录 1)

# 功能测试

### 预约与取消功能

运行./main test/reserve_cancel.txt，可以得到输出结果
这里的注释是不能在测试文件中出现的，否则会出现读取错误，这里是为了方便描述而写上的

```
10
10
3
0.1
agent 1
reserve 3
ticket 3
cancel 3
check_customer 1
reserve "B 2" Barry2 //预约超时测试
reserve "C 2" Barry1 //预约取消测试
end
agent 2
reserve 3
ticket 3
cancel 3
check_customer 1
ticket "B 2" Barry3
ticket "B 2" Barry3
ticket "B 2" Barry3
ticket "B 2" Barry3 //此时可以成功预约
end
agent 2
reserve 3
ticket 3
cancel 3
check_customer 1
cancel "B 2" Barry3 //取消不是自己的会报错
cancel "C 2" Barry1 //取消自己的
end
```

为了方便测试，使用了 6 秒作为最长预约时间，agent 2 在第四次请求会成功预约，前 3 次由于作为还在 Barry2 手上，所以不能买。详细测试结果见附录 2

### 购票与显示用户所有票功能

运行./main test/ticket_show.txt，可以得到输出结果

```
10
10
2
0.1
agent 1
reserve 3
ticket 3
cancel 3
check_customer 1
ticket "A 2" Barry1 //购票
ticket "B 2" Barry2 //购票
ticket "C" Barry3   //购买一排
end
agent 2
reserve 3
ticket 3
cancel 3
check_customer 1
show all Barry1
show all Barry1
show all Barry2
show all Barry2
show all Barry3
show all Barry3     //为了正确显示，最好查询两次，第一次可能由于agent2先运行，所以会显示没购买
end
```

购票功能比较简单，不过为了能够正确显示，做好查询两次以上，因为第一次可能会由于后面的进程先运行，导致显示没有成功购买，详细测试结果见附录 3

**摘要功能交易总结测试**

截取测试 2 的交易总结测试如下

```
==============summary Tick: 9==========================
           rows:10 cols:10

----------------Locked seats:12-------------------
(A  ,2  ,Barry1)
(B  ,2  ,Barry2) (C  ,1  ,Barry3) (C  ,2  ,Barry3)
(C  ,3  ,Barry3) (C  ,4  ,Barry3) (C  ,5  ,Barry3)
(C  ,6  ,Barry3) (C  ,7  ,Barry3) (C  ,8  ,Barry3)
(C  ,9  ,Barry3) (C  ,10 ,Barry3)
----------------Reserved seats:0---------------

----------------Available seats:88---------------
(A  ,1  )
(A  ,3  ) (A  ,4  ) (A  ,5  ) (A  ,6  ) (A  ,7  )
(A  ,8  ) (A  ,9  ) (A  ,10 ) (B  ,1  ) (B  ,3  )
(B  ,4  ) (B  ,5  ) (B  ,6  ) (B  ,7  ) (B  ,8  )
(B  ,9  ) (B  ,10 ) (D  ,1  ) (D  ,2  ) (D  ,3  )
(D  ,4  ) (D  ,5  ) (D  ,6  ) (D  ,7  ) (D  ,8  )
(D  ,9  ) (D  ,10 ) (E  ,1  ) (E  ,2  ) (E  ,3  )
(E  ,4  ) (E  ,5  ) (E  ,6  ) (E  ,7  ) (E  ,8  )
(E  ,9  ) (E  ,10 ) (F  ,1  ) (F  ,2  ) (F  ,3  )
(F  ,4  ) (F  ,5  ) (F  ,6  ) (F  ,7  ) (F  ,8  )
(F  ,9  ) (F  ,10 ) (G  ,1  ) (G  ,2  ) (G  ,3  )
(G  ,4  ) (G  ,5  ) (G  ,6  ) (G  ,7  ) (G  ,8  )
(G  ,9  ) (G  ,10 ) (H  ,1  ) (H  ,2  ) (H  ,3  )
(H  ,4  ) (H  ,5  ) (H  ,6  ) (H  ,7  ) (H  ,8  )
(H  ,9  ) (H  ,10 ) (I  ,1  ) (I  ,2  ) (I  ,3  )
(I  ,4  ) (I  ,5  ) (I  ,6  ) (I  ,7  ) (I  ,8  )
(I  ,9  ) (I  ,10 ) (J  ,1  ) (J  ,2  ) (J  ,3  )
(J  ,4  ) (J  ,5  ) (J  ,6  ) (J  ,7  ) (J  ,8  )
(J  ,9  ) (J  ,10 )
```

交易过后，显示当前空闲的座位，被预约的座位，被购买的座位，并且如果被预约或者购买，显示对应的用户

# 测试文件示例说明

在 test 文件夹下有三个测试文件，all_test.txt 测试文件是一个随便设置的测试文件，可以依据这个测试文件座位模板进行测试

在测试编写过程中，有以下注意点：

1. 用户名输入只能以字母开头并且内部只有字母和数字，否则这个交易会被跳过(利用正则表达式进行检查)
2. 所有交易的地址必须用引号括住，但是由于 show 操作只有 all 选项，所以不用加引号
3. 除了预约时间可以浮点数外，其它数字都是以整型变量传入，即使输入的数字是浮点数

# 总结

本项目的总结下来，有如下优点：

1. 代码框架设计规范，许多地方都尽可能地加上注释，方便后期修改代码以及增加代码地阅读性，尽可能地降低实体和功能的耦合程度，增加代码扩展性
2. 编写了 makefile，方便编译和调试
3. 直接使用共享内存和信号量的通信机制，灵活并且轻量，消息传输效率搞
4. 良好的格式化输出，方便查看运行结果，比如标识当前运行的时刻(单位秒)

本次项目使用了 Linux 的 IPC 机制进行音乐厅售票系统的设计，主要使用了信号量和共享内存以及进程扇框架，过程中也复习了许多操作系统方面的知识，虽然 C++内部提供了许多方便的线程库，但是自己动手直接使用系统调用可以更好地加深对课堂上学到的方法的理解。另外，这次项目代码量也不算小，过程中我的代码能力也得到了一定的提升。

总的来说，收获还是很大的。

# 附录

## 地址转换测试结果

==========agent: 1 Tick: 0==========

----------Barry buy----------

request position: A 2c //成功

----------result----------

Notice: all reserved only continue 60 seconds

success  seats:

(A,1)

(A,2)

agent 1 sleep for 2 seconds

==========good bye==========

==========agent: 2 Tick: 0==========

----------Barry buy----------

request position: D //成功

----------result----------

Notice: all reserved only continue 60 seconds

success  seats:

(D,1)

(D,2)

(D,3)

(D,4)

(D,5)

(D,6)

(D,7)

(D,8)

(D,9)

(D,10)

agent 2 sleep for 2 seconds

==========good bye==========

==========agent: 1 Tick: 2==========

----------Ellen buy----------

request position: B 2 //成功

----------result----------

Notice: all reserved only continue 60 seconds

success  seats:

(B,2)

agent 1 sleep for 2 seconds

==========good bye==========

==========agent: 2 Tick: 2==========

----------Barry buy----------

request position: E 2r //成功

----------result----------

Notice: all reserved only continue 60 seconds

success  seats:

(E,1)

(E,2)

(E,3)

(E,4)

(E,5)

(E,6)

(E,7)

(E,8)

(E,9)

(E,10)

(F,1)

(F,2)

(F,3)

(F,4)

(F,5)

(F,6)

(F,7)

(F,8)

(F,9)

(F,10)

agent 2 sleep for 2 seconds

==========good bye==========

==========agent: 1 Tick: 4==========

----------Barry buy----------

request position: C 2 5 //成功

----------result----------

Notice: all reserved only continue 60 seconds

successbuy: (C,2)

(C,3)

(C,4)

(C,5)

agent 1 sleep for 2 seconds

==========good bye==========

==========agent: 2 Tick: 4==========

----------Barry buy----------

request position: 6 //成功

----------result----------

Notice: all reserved only continue 60 seconds

success  seats:

(A,3)

(A,4)

(A,5)

(A,6)

(A,7)

(A,8)

agent 2 sleep for 2 seconds

==========good bye==========

==========agent: 1 Tick: 6==========

----------Barry buy----------

request position: F 11 //成功，无效地址输出 Error

----------result----------

Notice: all reserved only continue 60 seconds

success  seats:

Error: Invaild Position

agent 1 sleep for 2 seconds

==========good bye==========

==========agent: 2 Tick: 6==========

----------Barry buy----------

request position: K 5 ,无效地址输出 Error

----------result----------

Notice: all reserved only continue 60 seconds

success  seats:

Error: Invaild Position

agent 2 sleep for 2 seconds

==========good bye==========

# 预约功能测试结果

==========agent: 1 Tick: 0==========

----------Barry2 reserve----------

request position: B 2

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

(B,2)

agent 1 sleep for 3 seconds

==========good bye==========

==========agent: 2 Tick: 0==========

----------Barry3 buy----------

request position: B 2 //预约不成功，被占用

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

Fail: these seats are invaild or locked or reserved:

(B,2,reserved  座位被占用)

agent 2 sleep for 3 seconds

==========good bye==========

==========agent: 3 Tick: 0==========

----------Barry3 cancel----------

request position: B 2 //取消不成功，不属于自己的票

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

Fail: these seats are not belong to you:

(B,2,reserved  座位被占用)

agent 3 sleep for 3 seconds

==========good bye==========

==========agent: 1 Tick: 3==========

----------Barry1 reserve----------

request position: C 2

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

(C,2)

agent 1 sleep for 3 seconds

==========good bye==========

==========agent: 2 Tick: 3==========

----------Barry3 buy----------

request position: B 2

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

Fail: these seats are invaild or locked or reserved:

(B,2,reserved  座位被占用)

agent 2 sleep for 3 seconds

==========good bye==========

==========agent: 3 Tick: 3==========

----------Barry1 cancel----------

request position: C 2

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

(C,2)

agent 3 sleep for 3 seconds

==========good bye==========

==========agent: 2 Tick: 6==========

----------Barry3 buy----------

request position: B 2

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

Fail: these seats are invaild or locked or reserved:

(B,2,reserved  座位被占用)

agent 2 sleep for 3 seconds

==========good bye==========

==========agent: 2 Tick: 9==========

----------Barry3 buy----------

request position: B 2 //预约成功，此时票已经过期，可以预约

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

(B,2)

agent 2 sleep for 3 seconds

==========good bye==========

//输出摘要

==========summary Tick: 12==========

rows:10 cols:10

----------Locked seats:1----------

(B  ,2  ,Barry3) //结果正确

----------Reserved seats:0----------

----------Available seats:99----------

(A  ,1  )

(A  ,2  ) (A  ,3  ) (A  ,4  ) (A  ,5  ) (A  ,6  )

(A  ,7  ) (A  ,8  ) (A  ,9  ) (A  ,10 ) (B  ,1  )

(B  ,3  ) (B  ,4  ) (B  ,5  ) (B  ,6  ) (B  ,7  )

(B ,8 ) (B ,9 ) (B ,10 ) (C ,1 ) (C ,2 )

(C ,3 ) (C ,4 ) (C ,5 ) (C ,6 ) (C ,7 )

(C ,8 ) (C ,9 ) (C ,10 ) (D ,1 ) (D ,2 )

(D ,3 ) (D ,4 ) (D ,5 ) (D ,6 ) (D ,7 )

(D ,8 ) (D ,9 ) (D ,10 ) (E ,1 ) (E ,2 )

(E ,3 ) (E ,4 ) (E ,5 ) (E ,6 ) (E ,7 )

(E ,8 ) (E ,9 ) (E ,10 ) (F ,1 ) (F ,2 )

(F ,3 ) (F ,4 ) (F ,5 ) (F ,6 ) (F ,7 )

(F ,8 ) (F ,9 ) (F ,10 ) (G ,1 ) (G ,2 )

(G ,3 ) (G ,4 ) (G ,5 ) (G ,6 ) (G ,7 )

(G ,8 ) (G ,9 ) (G ,10 ) (H ,1 ) (H ,2 )

(H ,3 ) (H ,4 ) (H ,5 ) (H ,6 ) (H ,7 )

(H ,8 ) (H ,9 ) (H ,10 ) (I ,1 ) (I ,2 )

(I ,3 ) (I ,4 ) (I ,5 ) (I ,6 ) (I ,7 )

(I ,8 ) (I ,9 ) (I ,10 ) (J ,1 ) (J ,2 )

(J ,3 ) (J ,4 ) (J ,5 ) (J ,6 ) (J ,7 )

(J ,8 ) (J ,9 ) (J ,10 )

# 购票与显示功能测试结果

==========agent: 1 Tick: 0==========

----------Barry1 buy----------

request position: A 2

----------result----------

Notice: all reserved only continue 6 seconds

success seats:

(A,2)

agent 1 sleep for 3 seconds

==========good bye==========

==========agent: 2 Tick: 0==========

----------Barry1 show----------

request position:

----------result----------

reserved or buyed tickets:

(A,2,locked)

agent 2 sleep for 1 seconds

==========good bye==========

==========agent: 2 Tick: 1==========

----------Barry1 show----------

request position:

----------result----------

reserved or buyed tickets:

(A,2,locked)

agent 2 sleep for 1 seconds

==========good bye==========

==========agent: 2 Tick: 2==========

----------Barry2 show----------

request position:

----------result----------

reserved or buyed tickets:

NULL

agent 2 sleep for 1 seconds

==========good bye==========

==========agent: 1 Tick: 3==========

----------Barry2 buy----------

request position: B 2

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

(B,2)

agent 1 sleep for 3 seconds

==========good bye==========

==========agent: 2 Tick: 3==========

----------Barry3 show----------

request position:

----------result----------

reserved or buyed tickets:

NULL

agent 2 sleep for 1 seconds

==========good bye==========

==========agent: 1 Tick: 6==========

----------Barry3 buy----------

request position: C

----------result----------

Notice: all reserved only continue 6 seconds

success  seats:

(C,1)

(C,2)

(C,3)

(C,4)

(C,5)

(C,6)

(C,7)

(C,8)

(C,9)

(C,10)

agent 1 sleep for 3 seconds

==========good bye==========

==========summary Tick: 9==========

rows:10 cols:10

----------Locked seats:12----------

(A ,2 ,Barry1)

(B ,2 ,Barry2) (C ,1 ,Barry3) (C ,2 ,Barry3)

(C ,3 ,Barry3) (C ,4 ,Barry3) (C ,5 ,Barry3)

(C ,6 ,Barry3) (C ,7 ,Barry3) (C ,8 ,Barry3)

(C ,9 ,Barry3) (C ,10 ,Barry3)

----------Reserved seats:0----------

----------Available seats:88----------

(A ,1 )

(A ,3 ) (A ,4 ) (A ,5 ) (A ,6 ) (A ,7 )

(A ,8 ) (A ,9 ) (A ,10 ) (B ,1 ) (B ,3 )

(B ,4 ) (B ,5 ) (B ,6 ) (B ,7 ) (B ,8 )

(B ,9 ) (B ,10 ) (D ,1 ) (D ,2 ) (D ,3 )

(D ,4 ) (D ,5 ) (D ,6 ) (D ,7 ) (D ,8 )

(D ,9 ) (D ,10 ) (E ,1 ) (E ,2 ) (E ,3 )

(E ,4 ) (E ,5 ) (E ,6 ) (E ,7 ) (E ,8 )

(E ,9 ) (E ,10 ) (F ,1 ) (F ,2 ) (F ,3 )

(F ,4 ) (F ,5 ) (F ,6 ) (F ,7 ) (F ,8 )

(F ,9 ) (F ,10 ) (G ,1 ) (G ,2 ) (G ,3 )

(G ,4 ) (G ,5 ) (G ,6 ) (G ,7 ) (G ,8 )

(G ,9 ) (G ,10 ) (H ,1 ) (H ,2 ) (H ,3 )

(H ,4 ) (H ,5 ) (H ,6 ) (H ,7 ) (H ,8 )

(H ,9 ) (H ,10 ) (I ,1 ) (I ,2 ) (I ,3 )

(I ,4 ) (I ,5 ) (I ,6 ) (I ,7 ) (I ,8 )

(I ,9 ) (I ,10 ) (J ,1 ) (J ,2 ) (J ,3 )

(J ,4 ) (J ,5 ) (J ,6 ) (J ,7 ) (J ,8 )

(J ,9 ) (J ,10 )