朱胤恒|22920152203957 zhuyinheng@stu.xmu.edu.cn

基于 IPC 的进程通信

《Linux 内核分析》课程项目

摘要

本次作业的主题是利用 Linux IPC 的进程通信机制，通过模拟车辆在隧道中的行为达到练习使用 Linux IPC 接口进行进程间通信的目的。

本报告分为五个部分：

- 在第一部分介绍题目描述和项目输出；
- 在第二部分介绍在本项目实现过程中的使用到的 System V 接口函数；
- 在第三部分分析了题目描述的场景，设计了共享内存区中的并发冲突解决机制；
- 在第四部分介绍了在实现过程中的设计思路，通过设计隧道类、车辆类以及其他的辅助函数、控制函数实现模拟的过程；
- 在第五部分通过设计不同的输入场景，检验相应的功能是否正常工作。

本次作业除了报告之外，还提供源码、cmake 编译配置文件及编译使用方法、以及在第五部分设计的 5 组输入数据。

目录

1 项目描述

1.1 问题内容

该项目要求使用 Unix 高级 IPC 机制（包括信号量和共享内存）来实现进程同步。

实验要求实现一个多进程系统来模拟一个单向隧道内交通流量控制。出于安全原因，隧道内一次不能超过 N 辆车。入口处的交通灯控制车辆的进入，入口和出口处的车辆检测器则可以检测交通流量。当隧道已满时，到达的汽车将不允许进入隧道，直到隧道中有车辆离开。

隧道内的每辆车都可以访问和修改一个用以模拟隧道邮箱系统的共享内存段（可以看成是一个数组，访问操作操作包括：r 和 w），这样，隧道内的车辆就在进隧道后保持其手机通讯（隧道将阻塞手机信号）。隧道外的汽车则不需要访问该共享内存段。

每辆车都由一个 Linux 进程来模拟。当汽车在隧道内，它们的操作是允许并发的，并且必须与 Unix 高级 IPC 机制进行同步。如果一个给定邮箱尚未因写操作而锁定，则对它的读操作可以并行，但对同一邮箱的写操作则同一时刻只允许一个。

程序的输入包括隧道行程时间，每个存储段的大小以及车辆到达的顺序表，格式如下：

```
total_number_of_cars
maximum_number_of_cars_in_tunnel
tunnel_travel_time
total_number_of_mailboxes
memory_segment_size                    /* size of each mailbox */

car_no_1
operation duration mailbox-no
operation duration mailbox-no
  :
  :
end.

car_no_2
```

```
operation duration mailbox-no
operation duration mailbox-no
 :
 :
end.

  :
  :

car_no_LAST
operation duration mailbox-no
operation duration mailbox-no
 :
 :
end.
```

如果一个操作是写（w），则后面会跟随一个字符串，如下：

```
w 'hello' 20 5          /* write 'hello' into mailbox 5 with a
                           duration of 20 milliseconds */
```

如果一个操作是写（r），则对应邮箱里的目标信息将被读取并保存在车辆（也就是进程）的内存中以备过后打印，如下：

```
r 20 10 5               /* read 20 characters from mailbox 5 for 10
                           milliseconds */
```

每辆车针对每个邮箱都维护一个读指针，以防止其重复读取信息。如果没有更多字符需要读取，应打印出“邮箱末尾”消息。但是，每个邮箱只有一个写指针。

如果汽车在隧道外，则写入和读取操作都应打印如下消息：

```
car X is outside the tunnel writing (reading),
```

其中 X 表示车辆号。

除了上述的消息，每辆车的手机内存内容也应该打印出来。

1.2 示例输入

```
3
3
5
5
200
car_no_1
r 4 3 1
r 4 3 2
r 4 3 3
r 4 3 4
end.
car_no_2
r 4 3 1
r 4 3 2
r 4 3 3
r 4 3 4
end.
car_no_3
w aaaaaaa 1 1
w bbbbbbb 1 2
w ccccccc 1 3
r 4 3 1
r 4 3 2
r 4 3 3
r 4 3 4
end.
```

2 System V IPC

在本节中介绍在后续设计中使用到的 System V IPC 接口的使用方法，包括信号量、共享内存区、利用信号量构造锁。

值得注意的是，由于使用的是进程间共享的对象，因此相比于一般编程语言中定义的局部变量，不同之处在于可能需要在某个进程中创建、而在另一个进程中载入。这是由于在操作系统中，进程是资源分配的最小单位，跨进程的共享资源需要特别的说明。

2.1 使用信号量

在本项目中使用到的信号量的操作主要有 4 个：创建信号量、**初始化信号量**、载入信号量、删除信号量。

除此之外，对于信号量而言，还需要特别的初始化信号量，即将其值设为 1。

具体使用到的函数如下表：

功能	函数名	举例/说明
创建	`int semid = semget((key_t)key, number, IPC_CREAT|0666);`	● 创建键为 key，个数为 number 的信号量集合。 ● 当创建失败时，返回-1，否则返回信号量集合的编码。

初始化	`union semun sem_un;`
	`sem_un.val = 1;`
	`int rt=semctl(semid, i, SETVAL, sem_un);`
		● 将信号量集合编码为 semid 的第 i(从 0 开始计数)个信号量设置为 1。
		● 其中特别的是需要定义一个 union 变量传入参数。
		● 同样的，返回-1 表示设置失败。

载入	`int semid = semget((key_t)key, 0, IPC_CREAT);`	● 载入键为 key 的信号量集合 ● 返回-1 表示载入失败
删除	`int ret = semctl(semid, IPC_RMID, 0);`	● 删除信号量集合编码为 shemid 的信号量集合 ● 返回-1 表示删除失败

2.2 使用共享内存区

使用到的对于共享内存区的操作主要有：创建、载入、删除。

需要特别注意的是，**内存区的计数是以字节为单位的**。

具体使用的函数如下表：

功能	函数名	举例/说明

创建	`int shmid = shmget((key_t)key, number*sizeof(int), 0666 | IPC_CREAT);`	● 创建键为 key，长度为 number 的 int 数组。 ● 当创建失败时，返回-1，否则返回共享内存区的编码。
载入	`int shmid = shmget((key_t)key, 0, IPC_CREAT);` `void *shm = shmat(shmid, (void*)0, 0);`	● 载入键为 key 的内存区 ● 返回-1 表示载入失败，否则返回 void 指针
删除	`int ret = shmctl(shmid, IPC_RMID, 0);`	● 删除共享内存区编码为 shemid 的信号量集合 ● 返回-1 表示删除失败

2.3 构造锁

利用 System V 中对信号量的操作函数，可以构造 P、V 操作（如课件所述）。从而构造锁，进而可以构造临界区，对共享内存访问进行保护。

2.3.1 P

```
int P(int semid, int which)
{
    sembuf mysembuf;
    mysembuf.sem_num = which;
    mysembuf.sem_op = -1;
    mysembuf.sem_flg = SEM_UNDO;
    return semop(semid, &mysembuf, 1);
}
```

2.3.2 V

```
int V(int semid, int which)
{
    sembuf mysembuf;
    mysembuf.sem_num = which;
    mysembuf.sem_op = 1;
    mysembuf.sem_flg = SEM_UNDO;
    return semop(semid, &mysembuf, 1);
}
```

3 进程共享资源

在本节中，说明题目中需要利用共享资源处理的重要部分，设计安全有效的共享资源机制解决共享资源问题。

3.1 隧道满等待：锁与共享计数变量

注意到，题目中提到：

> 出于安全原因，隧道内一次不能超过 N 辆车。入口处的交通灯控制车辆的进入，入口和出口处的车辆检测器则可以检测交通流量。当隧道已满时，到达的汽车将不允许进入隧道，直到隧道中有车辆离开。

这是一个很经典的“满等待”的问题，可以通过锁（条件变量）结合共享计数变量实现。更为具体的，我们需要：

- 利用一个共享内存区的 int 变量，记录当前隧道内的车辆数
- 利用锁构造访问共享内存区内 int 变量的临界区，保证对计数的操作互斥
- 当一个进程期望进入隧道时：
    a) P 操作
    b) 判断共享内存区内的计数变量是否到达上限
        i. 若到达上限，则直接进行 V 操作。之后返回 a)
        ii. 若未到达上限，则计数器修改，进行 V 操作。之后进入隧道

值得注意的是，由于题目的额外要求：

> 如果汽车在隧道外，则写入和读取操作都应打印如下消息：
>
> car X is outside the tunnel writing (reading),

因此，实际的实现，比“满等待”问题更为复杂，对于这个问题需要在每次 PV 操作之后，释放一条最近的消息。

这样的操作是基于一种假设：

> 当一辆车进入隧道前进行读写操作一次消息不可中断。

这样的假设也比较合理，因为隧道内没有信号，需要在隧道外将一条消息发送或者接受完才能进入隧道。

3.2 邮箱读者写者问题冲突解决：读写锁

另一个核心的问题就是对邮箱读写的并发、互斥操作的冲突解决。其有如下要求：

- 对不同邮箱之间的操作不互相影响，可以并发访问
- 对同一邮箱，不同进程的读操作直接可以并发执行
- 对同一邮箱，不同进程的写操作互斥
- 对同一邮箱，读写操作互斥

因此，对于每个邮箱，需要通过三个锁，分别保证读写操作互斥、保护读进程数量计数、不同写操作互斥。

更为具体的过程如下：

- 对于写者而言：
    1. P(读写操作锁)
    2. P(写操作锁)
    3. 写入
    4. V(写操作锁)
    5. V(读写操作锁)
- 对于读者而言：
    1. P(读进程计数锁)
    2. 增加共享内存中的读进程数
    3. 如果当前进程是第一个读进程, P(读写操作锁)
    4. V(读进程计数锁)
    5. 读操作
    6. P(读进程计数锁)
    7. 减小共享内存中的读进程数
    8. 如果当前进程是最后一个进程，V(读写操作锁)
    9. V(读进程计数锁)

在具体实现中，由于还需要考虑到读者读取位置的记录等额外要求，因此还需要设置额外的独享指针记录每个邮箱读取的位置，防止出现重复读取的情况。

4 实现思路

这个项目的由两个类与一个控制脚本组成。其中，隧道类负责读取配置信息、创建共享内存区、存储隧道属性等；车辆类模拟车的行为；控制脚本负责创建进程。

除此之外，还有一些辅助类

4.1 隧道类

隧道类包含的成员变量/函数：

成员	含义
`vector<vector<r_message> > r;` `vector<vector<w_message> > w;`	全局的读写消息列表
`int total_number_of_mailboxes, memory_segment_size;` `int total_number_of_cars, maximum_number_of_cars_in_tunnel, tunnel_travel_time;`	全局的配置属性
`void setup_ipc();`	创建 IPC 共享变量、内存
`void clear_ipc();`	删除创建的 IPC 共享变量、内存
`void reset_env_ipc();`	删除所有的 IPC 共享变量、内存
`void read();`	按照输入格式，读取配置变量

定义：

```
#pragma once
#include"utils.h"
#include<vector>
#include<iostream>
using namespace std;
class tunnel
{
public:
    vector<vector<r_message> > r;
    vector<vector<w_message> > w;
    int total_number_of_mailboxes, memory_segment_size;
    int total_number_of_cars, maximum_number_of_cars_in_tunnel,
tunnel_travel_time;



    tunnel();
    void setup_ipc();
    void clear_ipc();
    void reset_env_ipc();
```

```
    void read();
    ~tunnel();
};
```

4.2 车辆类

其中涉及到的成员函数/变量：

成员	含义

`int idx_r, idx_w;`
`vector<r_message> my_r;`
`vector<w_message> my_w;`
	消息列表及其下表

`int id;`	进程号
`tunnel *T;`	绑定的隧道
`int *count,`	共享内存、隧道内的车辆数
`int *content,`	共享内存、多个邮箱
`int *reader_count,`	共享内村、每个邮箱的读者计数
`int *mailboxs_pointers;`	非共享内存、邮箱的读取位置
`int sem_mutex_reader_count,`	读者计数锁
`int sem_mutex_read_or_write,`	读者写者锁
`int sem_mutex_writer,`	写者锁
`int sem_waiting;`	满等待锁
`vector<vector<int> > readed_msg;`	手机内存
`vehicle(tunnel &T, int id);`	利用隧道 T 初始化车辆
`void waiting_and_in();`	等待进入隧道
`void leave();`	离开隧道
`void write_to_mailbox(w_message);`	写邮箱
`void read_from_mailbox(r_message,` `vector<int> &rt);`	读邮箱
`void run();`	车辆的行为逻辑

类的定义：

```
#pragma once
#include "tunnel.h"
#include "utils.h"
#include <vector>
#include <sys/types.h>
using namespace std;
class vehicle
{
```

```
private:
    int idx_r, idx_w;
    vector<r_message> my_r;
    vector<w_message> my_w;
    int id;
    tunnel *T;
    int *count,
        *content,
        *reader_count,
        *mailboxs_pointers;
    int sem_mutex_reader_count,
        sem_mutex_read_or_write,
        sem_mutex_writer,
        sem_waiting;
    vector<vector<int> > readed_msg;
public:
    vehicle(tunnel &T, int id);
    void waiting_and_in();
    void leave();
    void write_to_mailbox(w_message);
    void read_from_mailbox(r_message, vector<int> &rt);
    void run();
    ~vehicle();
};
```

4.3 控制脚本

创建隧道进程、创建车辆进程扇

```
#include <unistd.h>
#include "vehicle.h"
#include "tunnel.h"
using namespace std;
int main()
{
    freopen("/home/ubuntu/linuxproj/sample.in", "r", stdin);
    //freopen("/home/ubuntu/linuxproj/sample.out", "a", stdout);
    tunnel T;
    pid_t mypid;
    int vehicle_number=0;

    T.reset_env_ipc();
```

```
    T.read();
    T.setup_ipc();
    for (int i = 0; i < T.total_number_of_cars; i++)
    {
        mypid = fork();
        if (mypid == 0)
        {
            vehicle_number = i;
            printf("[Info]:  Car %d is coming\n", vehicle_number);
            break;
        }
    }

    if (mypid == 0)//child
    {
        //freopen("/home/ubuntu/linuxproj/sample.out", "a", stdout);
        vehicle my_car(T,vehicle_number);
        my_car.run();
    }

    return 0;
}
```

4.4 辅助类

4.4.1 消息类

表示读消息、写消息

```
class r_message
{
public:
    int length, mailbox_number, duration,id;
    r_message(int length, int mailbox_number, int duration,int id)
    {
        this->length = length;
        this->mailbox_number = mailbox_number;
        this->duration = duration;
        this->id = id;
    }
};
class w_message
{
public:
    string message;
```

```
    int mailbox_number, duration,id;
    w_message(string message, int mailbox_number, int duration,int id)
    {
        this->message = message;
        this->mailbox_number = mailbox_number;
        this->duration = duration;
        this->id = id;

    }
};
```

4.4.2 共享内存相关辅助函数

```
union semun {
    int              val;    /* Value for SETVAL */
    struct semid_ds *buf;    /* Buffer for IPC_STAT, IPC_SET */
    unsigned short  *array;  /* Array for GETALL, SETALL */
    struct seminfo  *__buf;  /* Buffer for IPC_INFO
                                (Linux-specific) */
};
void create_sem(int key, int number)
{
    int semid = semget((key_t)key, number, IPC_CREAT|0666);
    if (semid < 0)
    {
        perror("Semget error");
        exit(EXIT_FAILURE);
    }
    union semun sem_un;
    for(int i=0;i<number;i++)
    {
        sem_un.val = 1;
        int rt=semctl(semid, i, SETVAL, sem_un);
        if(rt<0)
        {
            perror("Semctl Set val error");
            exit(EXIT_FAILURE);
        }
    }
}
void create_sharedmem(int key, int number)
{
    int shmid = shmget((key_t)key, number*sizeof(int), 0666 |
IPC_CREAT);
    if (shmid == -1)
    {
```

```
        printf("shmget failed\n");
        exit(EXIT_FAILURE);
    }
}
void destroy_sharedmem(int key)
{
    int shmid = shmget((key_t)key, 0, IPC_CREAT);
    if (shmid == -1)
    {
            printf("Destroy null");
            exit(EXIT_FAILURE);
    }
    else
    {
        int ret = shmctl(shmid, IPC_RMID, 0);
        if (ret < 0)
        {
            printf("Destroy failed");
            exit(EXIT_FAILURE);
        }

    }
}
int load_sem(int key)
{
    int semid = semget((key_t)key, 0, IPC_CREAT);
    if (semid == -1)
    {
        perror("load error");
    }
    return semid;
}
int P(int semid, int which)
{
    sembuf mysembuf;
    mysembuf.sem_num = which;
    mysembuf.sem_op = -1;
    mysembuf.sem_flg = SEM_UNDO;
    return semop(semid, &mysembuf, 1);
}
int V(int semid, int which)
{
    sembuf mysembuf;
    mysembuf.sem_num = which;
```

```
    mysembuf.sem_op = 1;
    mysembuf.sem_flg = SEM_UNDO;
    return semop(semid, &mysembuf, 1);
}
void *load_mem(int key)
{
    int shmid = shmget((key_t)key, 0, IPC_CREAT);
    if (shmid == -1)
    {
        perror("shmget failed\n");
        exit(EXIT_FAILURE);
    }
    void *shm = shmat(shmid, (void*)0, 0);
    if (shm == (void*)-1)
    {
        perror("shmat failed\n");
        exit(EXIT_FAILURE);
    }
    return shm;
}
```

5 实验输入情况设计

针对实验的不同要求，设计了不同的实验输入，验证相关的功能.

5.1 隧道满等待

5.1.1 输入设置

输入设置:

变量	取值
车的总数	2
隧道最大车数目	1
隧道通过时间	5
邮箱数量	5
邮箱段大小	200

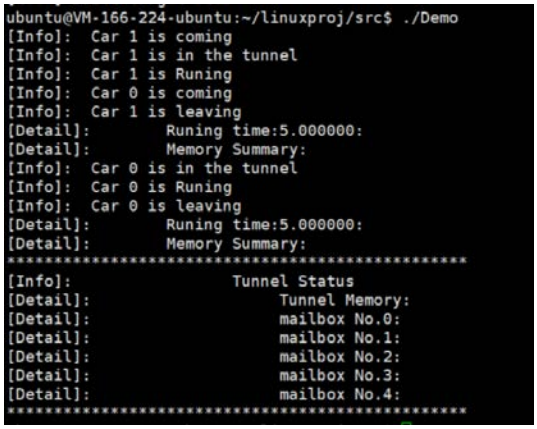
车辆行为:

car_no_1
end.
car_no_2
end.

5.1.2 预期结果

时间	车辆 0	车辆 1
1	进入隧道	隧道外等待
2		
3		
4		
5	退出隧道	进入隧道
6		
7		
8		
9		
10		退出隧道

5.1.3 实验结果



5.2 隧道外消息

5.2.1 输入设置

输入设置:

变量	取值
车的总数	2
隧道最大车数目	1
隧道通过时间	5
邮箱数量	5
邮箱段大小	200

车辆行为:

```
car_no_1
w aaa 2 1
w bbb 2 2
r 4 1 1
r 4 1 2
```

```
end.
car_no_2
w ccc 2 1
w ddd 2 2
r 4 1 1
r 4 1 2
end.
```

5.2.2 预期输出

时间	车辆 0	车辆 1
1	进入隧道 2 秒写操作(1/2)	隧道外等待 隧道外写操作(1/2)
2	2 秒写操作(2/2)	隧道外写操作(2/2)
3	2 秒写操作(1/2)	隧道外写操作(1/2)
4	2 秒写操作(2/2)	隧道外写操作(2/2)
5	2 秒读操作(1/1)读取内容:aaa***botton of mailbox*** 退出隧道	隧道外写操作(1/1) 进入隧道
6		
7		
8		
9		
10		退出隧道

5.2.3 实际输出

```
ubuntu@VM-166-224-ubuntu:~/linuxproj/src$ ./Demo
[Info]:  Car 1 is coming
[Info]:  Car 1 is in the tunnel
[Info]:  Car 0 is coming
[Info]:  Car 0 is outside the tunnel writing for 2 sec
[Info]:  Car 1 is Writing
[Detail]:        Write 2 sec in mailbox No. 1
[Detail]:        The content:ccc
[Info]:  Car 0 is outside the tunnel writing for 2 sec
[Info]:  Car 1 is Writing
[Detail]:        Write 2 sec in mailbox No. 2
[Detail]:        The content:ddd
[Info]:  Car 0 is outside the tunnel reading for 1 sec
[Info]:  Car 1 is Reading
[Detail]:        Read 1 sec in mailbox No. 1
[Detail]:        The content:ccc***mailbox botton***
[Info]:  Car 0 is outside the tunnel reading for 1 sec
[Info]:  Car 1 is Reading
[Detail]:        Read 1 sec in mailbox No. 2
[Detail]:        The content:ddd***mailbox botton***
[Info]:  Car 1 is leaving
[Detail]:        Runing time:6.000000:
[Detail]:        Memory Summary:
[Detail]:            Messge No. 0:ccc***mailbox botton***
[Detail]:            Messge No. 1:ddd***mailbox botton***
[Info]:  Car 0 is in the tunnel
[Info]:  Car 0 is Runing
[Info]:  Car 0 is leaving
[Detail]:        Runing time:5.000000:
[Detail]:        Memory Summary:
**********************************************
[Info]:                Tunnel Status
[Detail]:                   Tunnel Memory:
[Detail]:                   mailbox No.0:
[Detail]:                   mailbox No.1:ccc
[Detail]:                   mailbox No.2:ddd
[Detail]:                   mailbox No.3:
[Detail]:                   mailbox No.4:
**********************************************
```

5.3 隧道内并发访问不同邮箱

5.3.1 输入设置

输入设置:

变量	取值
车的总数	2
隧道最大车数目	2
隧道通过时间	5
邮箱数量	5
邮箱段大小	200

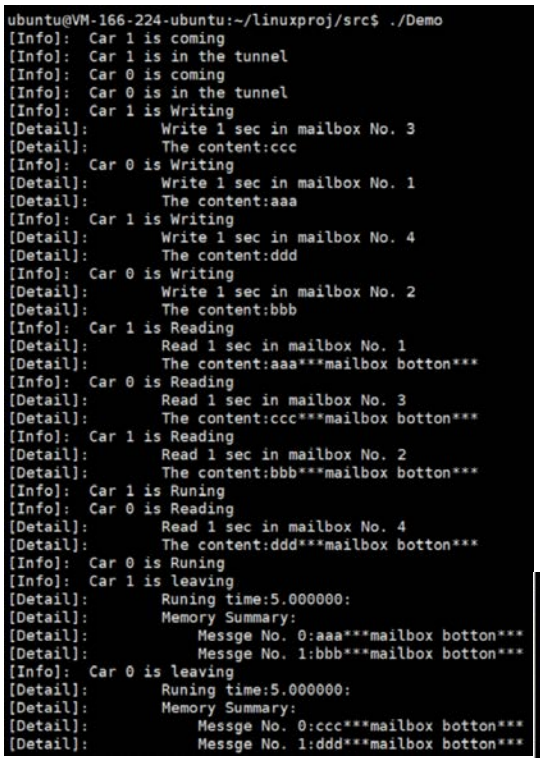
车辆行为:

```
car_no_1
w aaa 1 1
w bbb 1 2
r 4 1 3
r 4 1 4
end.
car_no_2
w ccc 1 3
w ddd 1 4
r 4 1 1
r 4 1 2
end.
```

5.3.2 预期输出

时间	车辆 0	车辆 1
1	进入隧道 1 秒写操作(1/1)	进入隧道 1 秒写操作(1/1)
2	1 秒写操作(1/1)	1 秒写操作(1/1)
3	1 秒读操作(1/1)	1 秒读操作(1/1)
4	1 秒读操作(1/1)	1 秒读操作(1/1)
5	退出隧道	退出隧道

5.3.3 实际输出



5.4 隧道内访问同一邮箱多读者并发，写者等待

5.4.1 输入设置

输入设置:

变量	取值
车的总数	3
隧道最大车数目	3
隧道通过时间	20
邮箱数量	5
邮箱段大小	200

车辆行为:

```
car_no_1
r 1 5 1
end.
car_no_2
r 1 3 1
r 1 7 1
end.
car_no_3
r 1 1 1
w ddd 1 1
```

```
end.
```

5.4.2 预期输出

时间	车辆 0	车辆 1	车辆 2
1	进入隧道 5 秒读操作(1/5)	进入隧道 3 秒读操作(1/3)	进入隧道 1 秒读操作(1/1)
2	5 秒读操作(2/5)	3 秒读操作(2/3)	等待
3	5 秒读操作(3/5)	3 秒读操作(3/3)	等待
4	5 秒读操作(4/5)	7 秒读操作(1/7)	等待
5	5 秒读操作(5/5)	7 秒读操作(2/7)	等待
6		7 秒读操作(3/7)	等待
7		7 秒读操作(4/7)	等待
8		7 秒读操作(5/7)	等待
9		7 秒读操作(6/7)	等待
10		7 秒读操作(7/7)	等待
11			1 秒写操作(1/1)
12			
…			
20	退出隧道	退出隧道	退出隧道

5.4.3 实际输出

```
ubuntu@VM-166-224-ubuntu:~/linuxproj/src$ ./Demo
[Info]:  Car 2 is coming
[Info]:  Car 2 is in the tunnel
[Info]:  Car 1 is coming
[Info]:  Car 1 is in the tunnel
[Info]:  Car 0 is coming
[Info]:  Car 0 is in the tunnel
[Info]:  Car 2 is Reading
[Detail]:        Read 1 sec in mailbox No. 1
[Detail]:        The content:***mailbox botton***
[Info]:  Car 1 is Reading
[Detail]:        Read 3 sec in mailbox No. 1
[Detail]:        The content:***mailbox botton***
[Info]:  Car 0 is Reading
[Detail]:        Read 5 sec in mailbox No. 1
[Detail]:        The content:***mailbox botton***
[Info]:  Car 0 is Runing
[Info]:  Car 1 is Reading
[Detail]:        Read 7 sec in mailbox No. 1
[Detail]:        The content:***mailbox botton***
[Info]:  Car 1 is Runing
[Info]:  Car 2 is Writing
[Detail]:        Write 1 sec in mailbox No. 1
[Detail]:        The content:ddd
[Info]:  Car 2 is Runing
[Info]:  Car 1 is leaving
[Detail]:        Runing time:20.000000:
[Detail]:        Memory Summary:
[Detail]:            Messge No. 0:***mailbox botton***
[Detail]:            Messge No. 1:***mailbox botton***
[Info]:  Car 2 is leaving
[Detail]:        Runing time:20.000000:
[Detail]:        Memory Summary:
[Detail]:            Messge No. 0:***mailbox botton***
[Info]:  Car 0 is leaving
[Detail]:        Runing time:20.000000:
[Detail]:        Memory Summary:
[Detail]:            Messge No. 0:***mailbox botton***
```

```
[Info]:  Car 0 is leaving
[Detail]:        Runing time:20.000000:
[Detail]:        Memory Summary:
[Detail]:            Messge No. 0:***mailbox botton***
*********************************************
[Info]:                Tunnel Status
[Detail]:                   Tunnel Memory:
[Detail]:                   mailbox No.0:
[Detail]:                   mailbox No.1:ddd
[Detail]:                   mailbox No.2:
[Detail]:                   mailbox No.3:
[Detail]:                   mailbox No.4:
*********************************************
```

5.5 隧道内访问同一邮箱写者等待

5.5.1 输入设置

输入设置:

变量	取值
车的总数	2
隧道最大车数目	2
隧道通过时间	5
邮箱数量	5
邮箱段大小	200

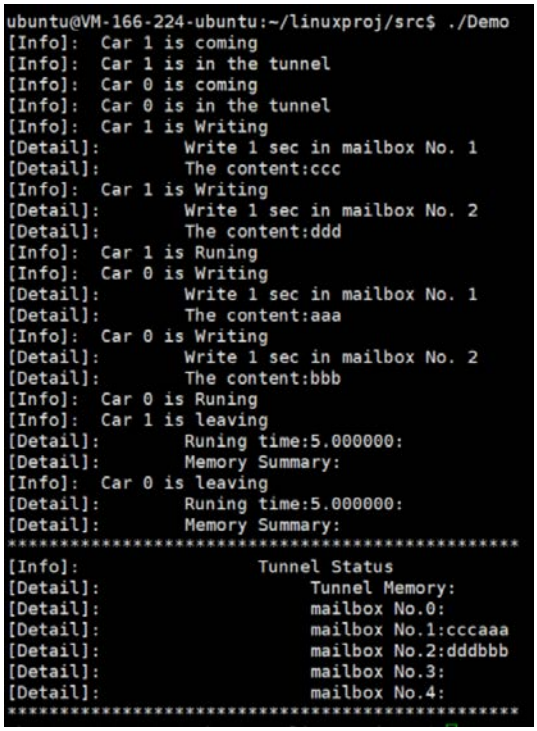
车辆行为:

```
car_no_1
w aaa 1 1
w bbb 1 2
end.
car_no_2
w ccc 1 1
w ddd 1 2
end.
```

5.5.2 预期输出

时间	车辆 0	车辆 1
1	进入隧道 1 秒写操作(1/1)	进入隧道 等待
2	1 秒写操作(1/1)	1 秒写操作(1/1)
3		1 秒读操作(1/1)
4		
5	退出隧道	退出隧道

5.5.3 实际输出


ubuntu@VM-166-224-ubuntu:~/linuxproj/src$ ./Demo
[Info]: Car 1 is coming
[Info]: Car 1 is in the tunnel
[Info]: Car 0 is coming
[Info]: Car 0 is in the tunnel
[Info]: Car 1 is Writing
[Detail]: Write 1 sec in mailbox No. 1
[Detail]: The content:ccc
[Info]: Car 1 is Writing
[Detail]: Write 1 sec in mailbox No. 2
[Detail]: The content:ddd
[Info]: Car 1 is Runing
[Info]: Car 0 is Writing
[Detail]: Write 1 sec in mailbox No. 1
[Detail]: The content:aaa
[Info]: Car 0 is Writing
[Detail]: Write 1 sec in mailbox No. 2
[Detail]: The content:bbb
[Info]: Car 0 is Runing
[Info]: Car 1 is leaving
[Detail]: Runing time:5.000000:
[Detail]: Memory Summary:
[Info]: Car 0 is leaving
[Detail]: Runing time:5.000000:
[Detail]: Memory Summary:
****
[Info]: Tunnel Status
[Detail]: Tunnel Memory:
[Detail]: mailbox No.0:
[Detail]: mailbox No.1:cccaaa
[Detail]: mailbox No.2:dddbbb
[Detail]: mailbox No.3:
[Detail]: mailbox No.4:
****