浙江大学实验报告

课程名称：操作系统分析及实验 实验类型：综合型/设计性

实验项目名称：实验 1 同步互斥与 Linux 内核模块

学生姓名：卢佳盈 专业：计算机科学与技术 学号：3180103570

电子邮件地址：ljy28501@163.com 手机：18868703211

实验日期：2020 年 11 月 20 日

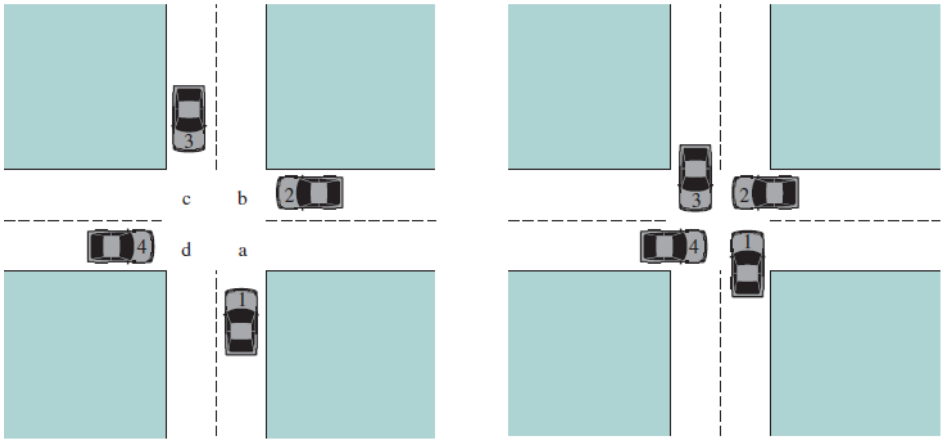
一、实验目的

- 学习使用 Linux 的系统调用和 pthread 线程库编写程序。
- 充分理解对共享变量的访问需要原子操作。
- 进一步理解、掌握操作系统进程和线程概念，进程或线程的同步与互斥。
- 学习编写多线程程序，掌握解决多线程的同步与互斥问题。
- 学习 Linux 模块的实现机理，掌握如何编写 Linux 模块。
- 通过对 Linux 系统中进程的遍历，进一步理解操作系统进程概念和进程结构。

二、实验内容

1. 编写一个Linux的内核模块，其功能是遍历操作系统所有进程。该内核模块输出系统中：每个进程的名字、进程pid、进程的状态、父进程的名字；以及统计系统中进程个数，包括统计系统中TASK_RUNNING、TASK_INTERRUPTIBLE、TASK_UNINTERRUPTIBLE、TASK_ZOMBIE、TASK_STOPPED等（还有其他状态）状态进程的个数。同时还需要编写一个用户态下执行的程序，显示内核模块输出的内容。

2. 有两条道路双向两个车道，即每条路每个方向只有一个车道，两条道路十字交叉。假设车辆只能向前直行，而不允许转弯和后退。如果有4辆车几乎同时到达这个十字路口，如图（a）所示；相互交叉地停下来，如图（b），此时4辆车都将不能继续向前，这是一个典型的死锁问题。从操作系统原理的资源分配观点，如果4辆车都想驶过十字路口，那么对资源的要求如下：
   - 向北行驶的车 1 需要象限 a 和 b；
   - 向西行驶的车 2 需要象限 b 和 c；
   - 向南行驶的车 3 需要象限 c 和 d；
   - 向东行驶的车 4 需要象限 d 和 a。



(a) Deadlock possible　　(b) Deadlock

我们要实现十字路口交通的车辆同步问题，防止汽车在经过十字路口时产生死锁和饥饿。在我们的系统中，东西南北各个方向不断地有车辆经过十字路口（注意：不只有4辆），同一个方向的车辆依次排队通过十字路口。按照交通规则是右边车辆优先通行，如图(a)中，若只有car1、car2、car3，那么车辆通过十字路口的顺序是car3->car2->car1。车辆通行总的规则：

1) 来自同一个方向多个车辆到达十字路口时，车辆靠右行驶，依次顺序通过；
2) 有多个方向的车辆同时到达十字路口时，按照右边车辆优先通行规则，除非该车在十字路口等待时收到一个立即通行的信号；
3) 避免产生死锁；
4) 避免产生饥饿；
5) 任何一个线程（车辆）不得采用单点调度策略；
6) 由于使用 AND 型信号量机制会使线程（车辆）并发度降低且引起不公平（部分线程饥饿），本题不得使用 AND 型信号量机制，即在上图中车辆不能要求同时满足两个象限才能顺利通过，如南方车辆不能同时判断 a 和 b 是否有空。

三、主要仪器设备（必填）

Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz

Linux version 5.9.1

Ubuntu 9.3.0-17

四、遍历操作系统所有进程

1. 编写目标：内核模块读取所有进程信息并输出到日志；用户态程序在日志中筛选模块加载信息，并将所有信息输出到屏幕

2. 设计思路：

命名遍历进程模块为 system_traveral，命名用户态输出模块为 user_print

【Part 1】system_traversal

system_tracersal.c

```
#include<linux/module.h>            //in order to create a module
#include<linux/kernel.h>            //in order to KERN_INFO
#include <linux/sched.h>            //in order to get task scheduling functions
#include <linux/sched/task.h>       //in order to use next_task
#include <linux/sched/signal.h>     //in order to use next_task

//Definition of global variables
int num_tot_proc=0;                 //total number of processors
int num_running_proc=0;             //number of running processors
int num_interruptible_proc=0;       //number of interruptible processors
int num_uninterruptible_proc=0;     //number of uninterruptible processors
int num_stopped_proc=0;             //number of stopped processors
int num_traced_proc=0;              //number of traced processors
int num_zombie_proc=0;              //number of zombie processors
int num_dead_proc=0;                //number of dead processors
int num_unknown=0;                  //number of unknown processors

//function to init the system_traversal module
int init_module(){
    struct task_struct *tmpTask;        //declare a pointer to tast_struct
    int tmpTask_state;                  //define an integer to store the temporary
state
    int tmpTask_exitState;              //define an interger to store the temporary
 exit state
    printk(KERN_INFO"Info of all processors\n");    //begin to traversal
    for(tmpTask=&init_task;(tmpTask=next_task(tmpTask))!=&init_task;){      //move
the pointer to the next task
        printk(KERN_INFO"Name:%s Pid:%d State:%ld ParName:%s\n",tmpTask->comm,tmpTa
sk->pid,tmpTask->state,tmpTask->real_parent->comm);  //print out the information of
 the task
        num_tot_proc++;                 //number of total processors added
        tmpTask_state=tmpTask->state;   //store the state of the task
        tmpTask_exitState=tmpTask->exit_state;  //store the exit state of the task
        if(tmpTask_exitState!=0){//the process has exited
            switch (tmpTask_state)      //classify the task by its state
            {
            case EXIT_ZOMBIE:num_zombie_proc++;break;   //number of zombie processo
rs added
            case EXIT_DEAD:num_dead_proc++;break;       //number of dead processors
 added
            default:break;
```

```
            }
        }else{//the process has not exited
            switch (tmpTask_state)      //classify the task by its state
            {
            case TASK_RUNNING:num_running_proc++;break;                     //numbe
r of running processors added
            case TASK_INTERRUPTIBLE:num_interruptible_proc++;break;         //numbe
r of interruptible processors added
            case TASK_UNINTERRUPTIBLE:num_uninterruptible_proc++;break;     //numbe
r of uninterruptible processors added
            case TASK_STOPPED:num_stopped_proc++;break;                     //numbe
r of stopped processors added
            case TASK_TRACED:num_traced_proc++;break;                       //numbe
r of traced processors added
            default:num_unknown++;break;                                    //numbe
r of unknown processors added
            }
        }
    }
    printk("******************Statistic INFO*********************\n");
    printk("Total tasks:%d\n",num_tot_proc);                            //print out
 the total number of processors
    printk("TASK_RUNNING:%d\n",num_running_proc);                       //print out
 the number of running processors
    printk("TASK_INTERRUPTIBLE:%d\n",num_interruptible_proc);           //print out
 the number of interruptible processors
    printk("TASK_UNINTERRUPTIBLE:%d\n",num_uninterruptible_proc);       //print out
 the number of uninterruptible processors
    printk("TASK_STOP:%d\n",num_stopped_proc);                          //print out
 the number of stopped processors
    printk("TASK_TRACED:%d\n",num_traced_proc);                         //print out
 the number of traced processors
    printk("EXIT_ZOMBIE:%d\n",num_zombie_proc);                         //print out
 the number of zombie processors
    printk("EXIT_DEAD:%d\n",num_dead_proc);                             //print out
 the number of dead processors
    printk("UNKNOWN:%d\n",num_unknown);                                 //print out
 the number of unknown processors
    return 0;
}

//function:clean_up module
void cleanup_module(){
    printk(KERN_INFO"end printing!!!!\n");
```

```
}
```

Makefile:

```
TARGET=system_traversal
KDIR=/usr/src/linux
PWD=$(shell pwd)
obj-m +=$(TARGET).o
default:
    make -C $(KDIR) M=$(PWD) modules
```

【Part 2】user_print

user_print.c

```
#include<stdio.h>
#include<string.h>
int main(){
    char info[1000]={0};        //store infomation read from log
    int file;                   //a variable to store if in the end of log
    FILE *fp=fopen("/var/log/kern.log","r");    //store the log file
    if(fp!=NULL){               //if open succesfully
        file=feof(fp);          //check if it is in the end of log
        while(!file){           //if not in the end
            memset(info,0,sizeof(info));    //initialize the space to store
            fgets(info,sizeof(info)-1,fp);  //read the log to the info
            printf("%s",info);              //print out
        }
        fclose(fp);
        printf("************END of LOG***************\n");
        return 0;               //end the programm
    }else{                      //if open unsuccessfully
        printf("Open log fail\n");
        return -1;              //return error
    }
}
```

【实验结果分析】

1. 编译 system_traversal.c，加载该模块

```
root@miracle-virtual-machine:~/osCode# cd system_traversal
root@miracle-virtual-machine:~/osCode/system_traversal# make
make -C /usr/src/linux M=/root/osCode/system_traversal modules
make[1]: 进入目录“/usr/src/linux-5.9.1”
root@miracle-virtual-machine:~/osCode/system_traversal# insmod system_traversal.ko
```

2. 显示日志

```
#dmesg
```

```
[ 2211.399024] Name:firefox Pid:3077 State:1 ParName:systemd
[ 2211.399025] Name:Privileged Cont Pid:3182 State:1 ParName:IPC Launch
[ 2211.399025] Name:Web Content Pid:3218 State:1 ParName:IPC Launch
[ 2211.399026] Name:WebExtensions Pid:3249 State:1 ParName:IPC Launch
[ 2211.399027] Name:Web Content Pid:3303 State:1 ParName:IPC Launch
[ 2211.399027] Name:kworker/0:3 Pid:3511 State:1026 ParName:kthreadd
[ 2211.399028] Name:kworker/u256:0 Pid:3648 State:1026 ParName:kthreadd
[ 2211.399029] Name:kworker/1:0 Pid:3761 State:1026 ParName:kthreadd
[ 2211.399029] Name:kworker/u256:1 Pid:3771 State:1026 ParName:kthreadd
[ 2211.399030] Name:kworker/0:0 Pid:3843 State:1026 ParName:kthreadd
[ 2211.399030] Name:kworker/1:2 Pid:3879 State:1026 ParName:kthreadd
[ 2211.399031] Name:kworker/0:1 Pid:3955 State:1026 ParName:kthreadd
[ 2211.399031] Name:insmod Pid:4304 State:0 ParName:bash
[ 2211.399043] ******************Statistic INFO*********************
[ 2211.399044] Total tasks:302
[ 2211.399044] TASK_RUNNING:4
[ 2211.399044] TASK_INTERRUPTIBLE:226
[ 2211.399045] TASK_UNINTERRUPTIBLE:0
[ 2211.399045] TASK_STOP:0
[ 2211.399045] TASK_TRACED:0
[ 2211.399045] EXIT_ZOMBIE:0
[ 2211.399045] EXIT_DEAD:0
[ 2211.399046] UNKNOWN:72
```

3.编译运行用户态程序 user_print.c

```
root@miracle-virtual-machine:~/osCode/user_print# cc user_print.c -O
user_print.c: In function 'main':
user_print.c:11:13: warning: ignoring return value of 'fgets', declared with attribute warn_unused_result [-Wunused-result]
           fgets(info,sizeof(info)-1,fp);
           ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
root@miracle-virtual-machine:~/osCode/user_print# ./user_print.out
Nov 22 11:14:22 miracle-virtual-machine kernel: [147424.990709] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
Nov 22 11:14:45 miracle-virtual-machine kernel: [147447.871130] watchdog: BUG: soft lockup - CPU#1 stuck for 22s! [gdbus:10637]
Nov 22 11:14:45 miracle-virtual-machine kernel: [147447.871165] watchdog: BUG: soft lockup - CPU#0 stuck for 22s! [gnome-shell:1580]
Nov 22 11:14:45 miracle-virtual-machine kernel: [147447.871166] Modules linked in: helloworld(OE) rfcomm
```

成功将所有信息打印到屏幕：

```
root@miracle-virtual-machine: ~/osCode/user_print
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398992] Name:gsd-rfkill Pid:1618 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398993] Name:gsd-screensaver Pid:1619 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398994] Name:gsd-disk-utilit Pid:1622 State:1 ParName:gnome-session-b
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398994] Name:gsd-sharing Pid:1625 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398994] Name:gsd-smartcard Pid:1629 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398995] Name:gsd-sound Pid:1636 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398996] Name:gsd-usb-protect Pid:1637 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398996] Name:gsd-wacom Pid:1645 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398997] Name:gsd-wwan Pid:1647 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398997] Name:gsd-xsettings Pid:1649 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398998] Name:gsd-printer Pid:1683 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.398999] Name:ibus-engine-lib Pid:1737 State:1 ParName:ibus-daemon
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399016] Name:update-notifier Pid:1857 State:1 ParName:gnome-session-b
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399017] Name:gvfsd-network Pid:1990 State:1 ParName:gvfsd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399018] Name:sh Pid:2008 State:1 ParName:gvfsd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399018] Name:gvfsd-admin Pid:2009 State:1 ParName:sh
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399019] Name:gvfsd-dnssd Pid:2068 State:1 ParName:gvfsd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399019] Name:vmware-vmblock- Pid:2577 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399020] Name:vmtoolsd Pid:2599 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399021] Name:VGAuthService Pid:2621 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399021] Name:ManagementAgent Pid:2685 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399022] Name:systemd-network Pid:2800 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399022] Name:gnome-terminal- Pid:2933 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399023] Name:bash Pid:2943 State:1 ParName:gnome-terminal-
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399023] Name:su Pid:2952 State:1 ParName:bash
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399024] Name:bash Pid:2955 State:1 ParName:su
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399024] Name:firefox Pid:3077 State:1 ParName:systemd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399025] Name:Privileged Cont Pid:3182 State:1 ParName:IPC Launch
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399025] Name:Web Content Pid:3218 State:1 ParName:IPC Launch
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399026] Name:WebExtensions Pid:3249 State:1 ParName:IPC Launch
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399027] Name:Web Content Pid:3303 State:1 ParName:IPC Launch
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399027] Name:kworker/0:3 Pid:3511 State:1026 ParName:kthreadd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399028] Name:kworker/u256:0 Pid:3648 State:1026 ParName:kthreadd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399029] Name:kworker/1:0 Pid:3761 State:1026 ParName:kthreadd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399029] Name:kworker/u256:1 Pid:3771 State:1026 ParName:kthreadd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399030] Name:kworker/0:0 Pid:3843 State:1026 ParName:kthreadd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399030] Name:kworker/1:2 Pid:3879 State:1026 ParName:kthreadd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399031] Name:kworker/0:1 Pid:3955 State:1026 ParName:kthreadd
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399031] Name:insmod Pid:4304 State:0 ParName:bash
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399043] ******************Statistic INFO*********************
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399044] Total tasks:302
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399044] TASK_RUNNING:4
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399044] TASK_INTERRUPTIBLE:226
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399045] TASK_UNINTERRUPTIBLE:0
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399045] TASK_STOP:0
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399045] TASK_TRACED:0
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399045] EXIT_ZOMBIE:0
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399045] EXIT_DEAD:0
Nov 22 14:12:35 miracle-virtual-machine kernel: [ 2211.399046] UNKNOWN:72
```

五、The BATMAN

【初始化定义】

1. 四个象限的互斥锁，实现每次只能有一个线程通过路口

```
pthread_mutex_t mutex_a;
pthread_mutex_t mutex_b;
pthread_mutex_t mutex_c;
pthread_mutex_t mutex_d;
```

2. 加锁后检查是否产生死锁

```
pthread_mutex_t mutex_deadlock;
```

唤醒死锁程序

```
pthread_cond_t cond_wake_deadlock;
```

唤醒右方车辆

```
pthread_cond_t cond_right_lock;
```

3. 到达四个方向 ready 队列的锁

```
pthread_mutex_t mutex_wait_east;
pthread_mutex_t mutex_wait_west;
pthread_mutex_t mutex_wait_north;
pthread_mutex_t mutex_wait_south;
```

4. 四个方向在第一路口等待的车辆的锁

```
pthread_mutex_t mutex_waiting_east;
pthread_mutex_t mutex_waiting_west;
pthread_mutex_t mutex_waiting_north;
pthread_mutex_t mutex_waiting_south;
```

5. 给同方向下一辆车与该方向左边的车发送信号

```
pthread_cond_t cond_west;
pthread_cond_t cond_east;
pthread_cond_t cond_north;
pthread_cond_t cond_south;
```

6. 给同方向等待队列中的车发送信号

```
pthread_cond_t firstEast;
pthread_cond_t firstWest;
pthread_cond_t firstSouth;
pthread_cond_t firstNorth;
```

7. 四个方向车队的队列结构

```
struct queueCar
{
    pthread_t thread[MAX];
   int id[MAX];
   int front;
    int rear;
    int current;
    int count;
    queueCar() {
        front = rear = count = 0;
    }
```

```
    void push(int n) {
        count++;
        rear = (rear + 1) % MAX;
        id[rear] = n;
    }
    void pop() {
        count--;
        front = (front + 1) % MAX;
        current= id[front];
    }
};
queueCar queueSouth;
queueCar queueEast;
queueCar queueNorth;
queueCar queueWest;
```

【处理输入】

处理输入的 n, w, s, e 四种字符，按输入顺序生成四个方向的车辆线程并通过锁和条件变量实现汽车通过路口的调度。

```
for (int i = 0; i < len; i++) {
        switch (s[i]) {
        case 'w': {
            is_west = true;
            queueWest.push(num[i]);
            car[tot_car++] = queueWest.thread[queueWest.front];
            pthread_create(&queueWest.thread[queueWest.front],
                NULL, car_from_west, NULL);
            break;
        }
        case 'e': {
            is_east = true;
            queueEast.push(num[i]);
            car[tot_car++] = queueEast.thread[queueEast.front];
            pthread_create(&queueEast.thread[queueEast.rear],
                NULL, car_from_east, NULL);
            break;
        }
        case 's': {
            is_south = true;
            queueSouth.push(num[i]);
            car[tot_car++] = queueSouth.thread[queueSouth.rear];
            pthread_create(&queueSouth.thread[queueSouth.rear],
                NULL, car_from_south, NULL);
            break;
```

```
        }
        case 'n': {
            is_north = true;
            queueNorth.push(num[i]);
            car[tot_car++] = queueNorth.thread[queueNorth.rear];
            pthread_create(&queueNorth.thread[queueNorth.rear],
                NULL, car_from_north, NULL);
            break;
        }
        }
    }
```

输出：

车辆到达路口：car %d from %s arrives at crossing
车辆车离开路口：car %d from %s leaving crossing
产生死锁，指示某一方向的车优先通过：DEADLOCK: car jam detected, signalling %s to go

【死锁检测】

为了避免饿死现象，需要先给左边的车发信号，让其通过，然后再给同方向的下一辆车发信号。所以在给同方向的下一辆车发信号之前，进行 usleep 操作，让其等待处于等待中唤醒每个队列最前方的车辆，接着根据 dir 变量判断指示最新到达路口的车辆先行。

```
void* check_dead_lock(void* arg) {
    usleep(4000);
    wakeupall();

    while(1){
        pthread_mutex_lock(&mutex_deadlock);
        pthread_cond_wait(&cond_wake_deadlock, &mutex_deadlock);
        is_deadlock = true;
        printf("DEADLOCK: car jam detected, signalling");
        switch (dir) {
        case north: {printf(" East "); pthread_cond_signal(&cond_east);
 break; }
        case east: {printf(" South "); pthread_cond_signal(&cond_south)
; break; }
        case west: {printf(" North "); pthread_cond_signal(&cond_north)
; break; }
        case south: {printf(" West "); pthread_cond_signal(&cond_west);
 break; }
        }
        printf("to go\n");
```

```
        pthread_mutex_unlock(&mutex_deadlock);
    }
}
```

【各方向线程处理】

以 south 方向为例

```
void* car_from_south(void* arg)
```

运行流程：

（1）加等待锁，申请路口互斥锁，记录当前方向

（2）检测是否产生死锁，若产生，发送信号量给死锁处理函数

```
if (sem == 0) {
    //indicate that a deadlock happened
    pthread_cond_signal(&cond_wake_deadlock);
    //wait until the deadlock is solved
    pthread_cond_wait(&cond_right_lock, &mutex_a);

    usleep(2000);
    pthread_mutex_lock(&mutex_b);
    pthread_mutex_unlock(&mutex_a);
    printf("car %d from South leaves at crossing\n", queueSouth.current
);
    is_south = false;
    //resource add 1.
    sem_post(&empty);
    usleep(2000);
    pthread_mutex_unlock(&mutex_b);
    wakeupall();
    return NULL;
}
```

（3）若未产生死锁，调度右方车辆限行

```
else if (is_east) {
    flag = true;
    pthread_cond_wait(&cond_south, &mutex_block_south);
    if (is_deadlock) {
        usleep(2000);
        pthread_mutex_lock(&mutex_b);
        pthread_mutex_unlock(&mutex_a);
        printf("car %d from South leaves at crossing\n", queueSouth.cur
rent);
        if (dir == west)pthread_cond_signal(&cond_right_lock);
        else pthread_cond_signal(&cond_west);
        is_south = false;
        sem_post(&empty);
        usleep(2000);
```

```
        pthread_mutex_unlock(&mutex_b);
        return NULL;
    }
}
```

（4）线程结束，释放锁和变量

【实验结果分析】

标准输入：nswewewn

输出结果：

在标准 nswewewn 输入测试中，会产生一次死锁，北南西东四个方向的车同时到达路口，检测到死锁后，让西方向的车先开走。之后到达的车进行调度时不会再出现死锁，每次都让右侧的车先走，然后是下一个方向路口的车，同方向的下一辆车会在其他方向都走一辆车之后再走，避免产生饥饿问题。

六、问题解答

【Makefile】编写

在编写 Makefile 之前对 Linux 下 make 工具及 Makefile 文件规则进行了一定的了解，但由于忽略了 Linux 内核实际路径，根据 《边干边学——Linux 内核指导》一书中的 Makefile 示范代码初次编写的 Makefile 并不能成功执行。经检查，实验所用的 Linux 内核路径为： ：/usr/src/linux，修改后成功执行。还需注意 Makefile 不能写为 makefile ，否则会报错。

【user_print】

在编写 user_print.cpp 中文件打开代码时，代码写为：fs.open(file, ios::in)。在 Windows 环境下，这样写是没有问题的，可以正确编译，但在 Linux 下使用 g++却会报错：no matching function for call to std::basic_ifstream<char>::open(const string&)’ 查阅官方文档发现，这里出错的原因是，open 函数的第一个参数应该是“要打开的文件的文件名的字符串”，但 linux forums 中写道：C++ ofstream::open won’t accept string as

a filename，Linux 下会出问题原因就在于 string 不能直接作为 fstream 的参数直接传入，因此把代码修改为：

```
fs.open(file.c_str(), ios::in);
```

修改后可以成功编译。

【检测死锁】

解决方法：

给每个路口增加了变量和互斥锁，来标记当前路口是否有车到达且只能有一辆正在等待通过路口，其余的在 ready 队列中等待，赋予其他的锁和变量。

【饿死】

为了避免饿死现象，需要先给左边的车发信号，让其通过，然后再给同方向的下一辆车发信号。解决方法：在给同方向的下一辆车发信号之前，进行 usleep 操作，让其等待后再接受信号进行同方向车辆的放行。

七、讨论、心得

本实验除了帮助我们理解 task_struct 之外还帮助我们掌握了如何编写、加载一个内核模块，通过解决实际问题提高了对 Linux 基本命令的熟悉程度，为后续的实验打下了基础。通过对 BATMAN 代码的编写，我对于多线程程序的编写有了更加清楚的认识，之后的优化可以考虑使用数组方法统筹四个方向的进程锁和功能函数，减少代码量。本次实验帮助我们更深入地理解了多线程中互斥锁和变量的使用，感受了 Linux 下编译自己的程序的乐趣，实践中能更好地理解课堂中所学到的知识，在对 Linux 系统的操作中进一步探索操作系统的原理和奥妙。