浙江大学实验报告

课程名称：操作系统 实验类型：综合型/设计性

实验项目名称：实验一 同步互斥和 Linux 内核模块

学生姓名：钱旭峰 专业：计算机科学 学号：3140102491

电子邮件地址：17816862315@163.com 手机：17816862315

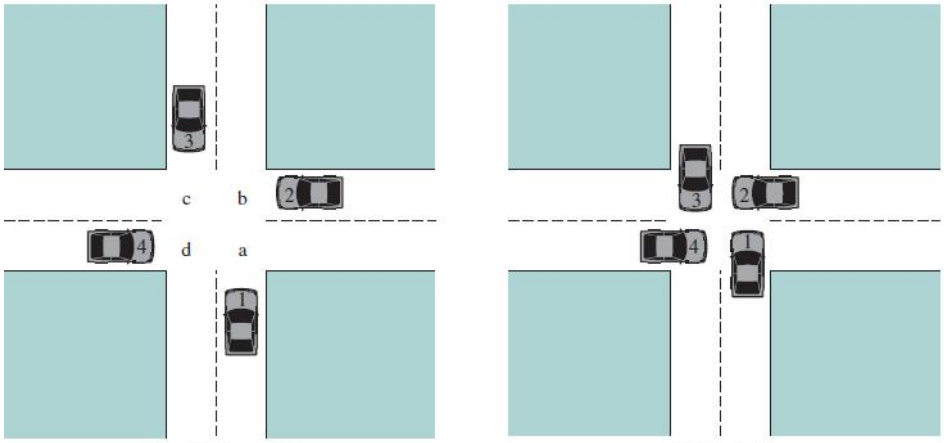
实验日期：2016 年 12 月 7 日

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. 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。

2.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。

三、主要仪器设备（必填）

华硕 X450V，unbantu 操作系统， 虚拟机 2G 内存

四、操作方法和实验步骤

第一部分（具体代码见 deadlock.c 文件）：

首先我分析了许多种情况，定义了一些变量和相应的锁

程序中一共有两种锁

1：通行锁（分为四个方向）

2：first position lock（即每个路口的第一个位置的占有锁）
一开始我会先输入所有需要通行的车辆，之后根据每辆车的方向，在每个方向上为每辆车排序，比如北方第一辆车到北方第 N 辆车，和西方第一辆车到西方第 N 辆车，每辆车为一个线程

之后我分为以下几种情况：
当某个方向的车（以北方为例）来的时候，他会先等待并判断他的右手边（西边）有没有车，若右手边有车，则等待右手边的车先过，当西边的那辆车通过之后会判断多种情况来发信号：
1.若西边没有车且北边还有车则只给北边发通行信号
2.若西边有车且北边没有车则他会判断南边有没有车，若没有则给西边发通行信号
3.若两边都有车则会判断 westToNorth 这个变量是否为 0，若为 0 则发出让北边车辆通行的信号，若为 1 则他会判断南边有没有车，若没有则给西边发通行信号这样做是为了保证当西边有车通行时，北边的车能够与西边的车轮流通行，不导致饥饿

当一辆车通过时，他会给该方向下一辆车发出 first position lock 的信号，使该方向下一辆车占有 first position，当一辆车占有 first position 之后，他就会等待放行信号

所以一般情况下一个方向的通信信号由其右手边发出，若其右手边没有车，则由该方向前一辆车为其发出通行信号

我会维护一个 wait 变量来记录等待的车辆数，当等待的车辆数到达 4 时，即判断为死锁，这时候就让北方的车先走。
第二部分：
我从 init_task 开始用 list_for_each(pos, &(initTask->tasks))遍历所有进程

```
list_for_each(pos, &(initTask->tasks))
{
    p=list_entry(pos, struct task_struct, tasks);
    total++;
    printk("%s pid:%d parent: %s parent's pid: %d ",p->comm,p->pid,p->real_parent->comm,p->real_parent->pid);

    switch(p->state)
    {
        case TASK_RUNNING:printk("state: RUNNING");running++;break;
        case TASK_INTERRUPTIBLE:printk("state: INTERRUPTIBLE");interruptible++;break;
        case TASK_UNINTERRUPTIBLE:printk("state: UNINTERRUPTIBLE");uninterruptible++;break;
        case TASK_TRACED:printk("state: TRACED");traced++;break;
        case TASK_STOPPED:printk("state: STOPPED");stopped++;break;
        case EXIT_ZOMBIE:zombie++;printk("exit_state: ZOMBIE");break;
        case EXIT_DEAD:dead++;printk("exit_state: DEAD");break;
        default:printk("state: UNKNOWN");unknown++;break;
    }

    printk("\n");
}
```

之后再输出到 kern.log 文件中。最后由 printk.c（由我自己编写）输出到控制台

五、实验结果和分析

第一部分：

```
anthony@anthony-virtual-machine:~$ ./deadlock
wesnnswensnwe
the car 1 from North arrives at intersection
the car 1 from South arrives at intersection
the car 1 from East arrives at intersection
the car 1 from West arrives at intersection
DEADLOCK FOUND,the car from north go first
the car 1 from North have left, which is the 1 car
the car 1 from East have left, which is the 2 car
the car 1 from South have left, which is the 3 car
the car 1 from West have left, which is the 4 car
the car 2 from South arrives at intersection
the car 2 from East arrives at intersection
the car 2 from North arrives at intersection
the car 2 from West arrives at intersection
DEADLOCK FOUND,the car from north go first
the car 2 from North have left, which is the 5 car
the car 2 from East have left, which is the 6 car
the car 2 from South have left, which is the 7 car
the car 2 from West have left, which is the 8 car
the car 3 from North arrives at intersection
the car 3 from South arrives at intersection
the car 3 from West arrives at intersection
the car 3 from East arrives at intersection
DEADLOCK FOUND,the car from north go first
the car 3 from North have left, which is the 9 car
the car 3 from East have left, which is the 10 car
the car 3 from South have left, which is the 11 car
the car 3 from West have left, which is the 12 car
the car 4 from North arrives at intersection
the car 4 from North have left, which is the 13 car
anthony@anthony-virtual-machine:~$
```

```
anthony@anthony-virtual-machine:~$ ./deadlock
weswesw
the car 1 from South arrives at intersection
the car 1 from East arrives at intersection
the car 1 from East have left, which is the 1 car
the car 1 from South have left, which is the 2 car
the car 1 from West arrives at intersection
the car 2 from South arrives at intersection
the car 2 from East arrives at intersection
the car 2 from East have left, which is the 3 car
the car 2 from South have left, which is the 4 car
the car 1 from West have left, which is the 5 car
the car 2 from West arrives at intersection
the car 2 from West have left, which is the 6 car
the car 3 from West arrives at intersection
the car 3 from West have left, which is the 7 car
```

第二部分：

```
state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875760] update-notifier pid:5618 parent: gnome-session-b parent's
pid: 5111 state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875763] gvfsd-dnssd pid:5632 parent: upstart parent's pid: 4902
state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875765] notify-osd pid:5655 parent: upstart parent's pid: 4902
state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875768] deja-dup-monito pid:5735 parent: gnome-session-b parent's
pid: 5111 state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875771] kworker/u256:0 pid:6549 parent: kthreadd parent's pid: 2
state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875773] kworker/0:0 pid:6603 parent: kthreadd parent's pid: 2
state: INTERRUPTIBLE
文件      :48 anthony-virtual-machine kernel: [ 7589.875776] dhclient pid:6774 parent: NetworkManager parent's pid:
4134 state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875778] kworker/0:2 pid:6873 parent: kthreadd parent's pid: 2
state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875781] update-manager pid:7121 parent: upstart parent's pid:
4902 state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875783] unity-control-c pid:7171 parent: upstart parent's pid:
4902 state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875786] unity-scope-hom pid:7200 parent: upstart parent's pid:
4902 state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875789] unity-scope-loa pid:7210 parent: upstart parent's pid:
4902 state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875791] unity-files-dae pid:7212 parent: upstart parent's pid:
4902 state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875793] kworker/0:1 pid:7249 parent: kthreadd parent's pid: 2
state: INTERRUPTIBLE
Dec 14 13:03:48 anthony-virtual-machine kernel: [ 7589.875796] kworker/u256:1 pid:7250 parent: kthreadd parent's pid: 2
```

```
anthony@anthony-virtual-machine:~$ sudo /sbin/insmod ProcessTraverse.ko
anthony@anthony-virtual-machine:~$ ps ax|grep ProcessTraverse
  7844 pts/4    S+     0:00 grep --color=auto ProcessTraverse
anthony@anthony-virtual-machine:~$ ./printk
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737142] kworker/0:1 pid:7780 parent: kthreadd parent's pid: 2 sta
te: INTERRUPTIBLE
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737143] sudo pid:7838 parent: bash parent's pid: 5350 state: INTE
RRUPTIBLE
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737144] insmod pid:7839 parent: sudo parent's pid: 7838 state: RU
NNING
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737145] total number of process is 255
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737146] TASK_RUNNING:5
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737146] TASK_INTERRUPTIBLE:250
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737147] TASK_UNONTERRUPTIBLE:0
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737148] TASK_STOPPED:0
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737148] TASK_TRACED:0
Nov 21 19:52:00 anthony-virtual-machine kernel: [11673.737149] TASK_UNKNOWN:0
anthony@anthony-virtual-machine:~$ clear
```

六、讨论、心得

在解决饥饿的问题上让我思考了很长时间，从开始到结束一共想了很多种方案，直到最后才得出一个令人满意的答案。一开始采用睡眠的方法，之后发现这样并不是真正的解决问题，之后就采用一个变量来记录两边的通行情况，来使两边能够实现轮流通行，最后才解决了饥饿的问题。

在第二个部分中，我了解了进程的状态，搜了比较久的资料，终于把所有进程的状态和父进程之类的输出了