

浙江大学实验报告

课程名称: 操作系统 实验类型: 综合型
实验项目名称: 同步互斥和 Linux 内核模块
学生姓名: 张佳瑶 学号: 3170103240
电子邮件地址: 1531077171@qq.com
实验日期: 2019 年 10 月 23 日

一、实验环境

处理器: 2.4GHz Intel Core i5

macOS 10.14.5

Linux version 5.0.0-29-generic (bulld@lgw01-amd64-039) (gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1))

处理器: Intel® Core™ i7-6700HQ CPU @ 2.60GHz

Windows10

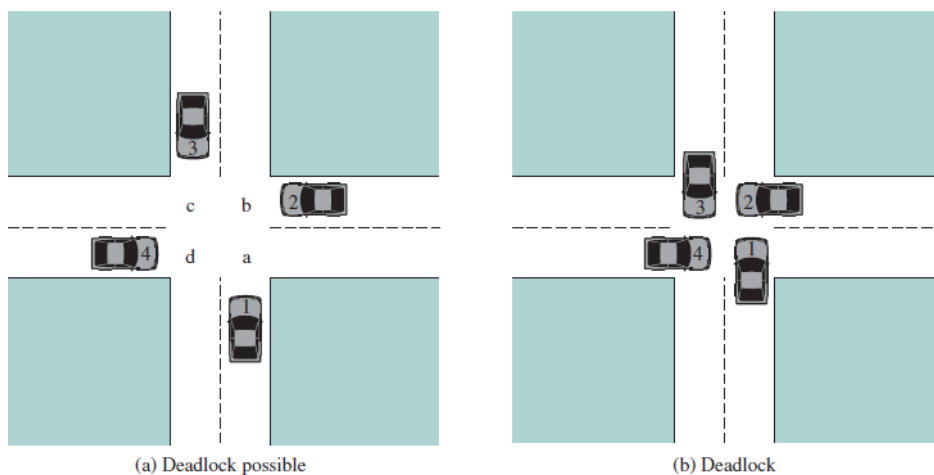
Linux version 5.0.0-29-generic (bulld@lgw01-amd64-039) (gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1))

二、实验内容和结果及分析

1.1 题目

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。



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。

编写程序实现避免产生死锁和饥饿的车辆通过十字路口方案，并给出详细的设计方案，程序中要有详细的注释（每三行代码必须要有注释）。

1.2 设计文档

1.2.1 pthread.h

1) 线程控制方面的函数有：pthreadattrinit、pthreadcreate、pthreadjoin、pthread_exit

函数	功能
pthreadattrinit	初始化一个线程对象的属性

pthread_create 四个参数：第一个参数是指向线程标识符的指针，**type:** **pthread_t***；第二个参数设置线程属性，**NULL** 则使用默认属性；第三个参数是线程运行函数的起始地址，**type:** **(void*)(*)(void*)**；第四个参数是运行函数的参数，**type:** **void***

pthread_join 用来等待一个线程的结束，线程间同步的操作，共两个参数：第一个参数为线程标识符，即线程 **id**，**type:** **pthread_t**；第二个参数 **retval** 为用户定义的指针，用来存储线程的返回值，**type: void****

pthread_exit 显式退出进程

2) 互斥锁机制函数：**pthreadmutexinit**、**pthreadmutexlock**、**pthreadmutexunlock**、**pthreadmutexdestroy**

互斥锁保证只有一个进程访问该对象。

锁类型结构 **pthreadmutex**

函数	功能
pthreadmutexinit	在 C 语言的多线程编程中，实现互斥锁的初始化
pthreadmutexlock	获取互斥锁
pthreadmutexunlock	释放互斥锁
pthreadmutexdestroy	注销一个互斥锁，参数为 pthreadmutex_t mutex

1.2.2 流程说明

1. 记录每一辆车的 **id** 和方向，将每一辆车加入到对应方向的车队中等待行驶。创建一个死锁检测线程，检测到死锁，就释放信号让北方车行驶。
2. 为每一辆车创建一个线程。
3. 判断线程对应的车是否为等待车队中的第一辆车，若是，则进入 4；若否，则循环等待。
4. 判断这个方向上是否还有车在行驶，若是，则等待条件变量唤醒；若否，则进入 5。
5. 该车可以到达路口，获得这个方向上的互斥锁。
6. 判断当前右方向是否有车等待，若是，等待可行驶条件变量；若否，进入 7。
7. 行驶当前的车，打印行驶信息。
8. 获得等待车队的互斥锁，将当前车移除等待队列，释放当前车队下一辆车可行驶的条件变量。
9. 所有线程结束后，释放所有的条件变量和互斥量

1.2.3 基本变量和函数声明

```
pthread_cond_t queue[4];           //各个方向车辆队列的条件变量
pthread_mutex_t lock[4];          //各个方向条件变量互斥锁
pthread_cond_t next[4];           //各个方向设置下次通行车辆的条件变量
pthread_mutex_t cross;             //路口通行互斥量
pthread_mutex_t wait[4];          //设置各个方向上等待队列的互斥量
struct carNode car[MAX];          //车量队列
struct carQueue waitCarQueue[4];  //各个方向等待的车的队列
int isCar[4] = {0};               //各个方向上的路口是否有车正在通行
void *carCross(void *car);         //车辆通过线程
void *checkDeadlock(void *item);  //死锁检测线程
```

1.3 程序运行结果截图

编译程序: g++ final.cpp -o thread -lpthread

运行程序: ./thread [车队]

```
zjy@ubuntu:~/Desktop/3$ ./a.out nswenswe
car 4 from East arrives at crossing
car 3 from West arrives at crossing
car 2 from South arrives at crossing
car 1 from North arrives at crossing
DEADLOCK: car jam detected, signalling North to go
car 1 from North leaving crossing
car 3 from West leaving crossing
car 2 from South leaving crossing
car 4 from East leaving crossing
car 6 from South arrives at crossing
car 7 from West arrives at crossing
car 5 from North arrives at crossing
car 8 from East arrives at crossing
^[^ADEADLOCK: car jam detected, signalling North to go
car 5 from North leaving crossing
car 7 from West leaving crossing
car 6 from South leaving crossing
car 8 from East leaving crossing
```

```
zjy@ubuntu:~/Desktop/3$ ./a.out nsewwewn
car 4 from West arrives at crossing
car 3 from East arrives at crossing
car 2 from South arrives at crossing
car 1 from North arrives at crossing
DEADLOCK: car jam detected, signalling North to go
car 1 from North leaving crossing
car 4 from West leaving crossing
car 2 from South leaving crossing
car 3 from East leaving crossing
car 8 from North arrives at crossing
car 5 from West arrives at crossing
car 6 from East arrives at crossing
car 6 from East leaving crossing
car 8 from North leaving crossing
car 5 from West leaving crossing
car 7 from West arrives at crossing
car 7 from West leaving crossing
```

```

zjy@ubuntu:~/Desktop/3$ ./a.out nsew
car 4 from West arrives at crossing
car 3 from East arrives at crossing
car 2 from South arrives at crossing
car 1 from North arrives at crossing
DEADLOCK: car jam detected, signalling North to go
car 1 from North leaving crossing
car 4 from West leaving crossing
car 2 from South leaving crossing
car 3 from East leaving crossing

```

1.4 结果分析

当少于四辆车到达十字路口时，不会发生死锁。

当有四辆车到达十字路口时，会发生死锁，释放北方车辆先行，解除死锁，其他方向车辆接着行驶。

```

zjy@ubuntu:~/Desktop/3$ ./a.out nsew
car 4 from West arrives at crossing
car 3 from East arrives at crossing
car 2 from South arrives at crossing
car 1 from North arrives at crossing
DEADLOCK: car jam detected, signalling North to go
car 1 from North leaving crossing
car 4 from West leaving crossing
car 2 from South leaving crossing
car 3 from East leaving crossing

```

为例：

释放 north 先行信号。1 号车最先行驶。1 号车右手边的 4 号车接着行驶。4 号车右手边的 2 号车接着行驶。最后 3 号车行驶。

1.5 源程序

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

#define MAX 100
#define NORTH 0
#define EAST 1
#define SOUTH 2
#define WEST 3

//一辆汽车
struct carNode
{
    int id;
    int direction;
};
//一个汽车队列
struct carQueue
{
    int size; //队列中车总数

```

```

    int head; //第一辆车
    int tail; //最后一辆车
    char direction[6];
    int cars[MAX];
};
pthread_cond_t queue[4]; //各个方向车辆队列的条件变量
pthread_mutex_t lock[4]; //各个方向条件变量互斥锁
pthread_cond_t next[4]; //各个方向设置下次通行车辆的条件变量
pthread_mutex_t cross; //路口通行互斥量
pthread_mutex_t wait[4]; //设置各个方向上等待队列的互斥量
pthread_cond_t deadlockCond; //死锁条件变量
pthread_mutex_t deadlockLock; //检测死锁互斥量
struct carNode car[MAX]; //车量队列
struct carQueue waitCarQueue[4]; //各个方向等待的车的队列
int isWait[4] = {0}; //各个方向上是否有车等待
int isCar[4] = {0}; //各个方向上的路口是否有车正在通行

void *carCross(void *car);

int main(int argc, char const *argv[])
{
    //初始化条件变量和互斥量
    for (int i = 0; i < 4; i++)
    {
        pthread_cond_init(&queue[i], NULL);
        pthread_cond_init(&next[i], NULL);
        pthread_mutex_init(&lock[i], NULL);
    }
    pthread_cond_init(&deadlockCond, NULL);
    pthread_mutex_init(&deadlockLock, NULL);

    //初始化等待队列
    for (int i = 0; i < 4; i++)
    {
        waitCarQueue[i].head = 0;
        waitCarQueue[i].tail = 0;
        waitCarQueue[i].size = 0;
    }

    //创建和总车数相等的线程
    pthread_t carThread[strlen(argv[1])];
    //标记好四个车队的方向
    strcpy(waitCarQueue[NORTH].direction, "North");
    strcpy(waitCarQueue[EAST].direction, "East");

```

```

strcpy(waitCarQueue[SOUTH].direction, "South");
strcpy(waitCarQueue[WEST].direction, "West");
//获得四个方向的车队
for (int i = 0; i < strlen(argv[1]); i++)
{
    switch (argv[1][i])
    {
        case 'n':
            //北方
            car[i].id = i + 1;
            car[i].direction = 0;
            //加入北方车队
            waitCarQueue[NORTH].size++;
            waitCarQueue[NORTH].cars[waitCarQueue[NORTH].tail++] = i
+ 1;

            break;
        case 'e':
            //东方
            car[i].id = i + 1;
            car[i].direction = 1;
            //加入东方车队
            waitCarQueue[EAST].size++;
            waitCarQueue[EAST].cars[waitCarQueue[EAST].tail++] = i +
1;

            break;
        case 's':
            //南方
            car[i].id = i + 1;
            car[i].direction = 2;
            //加入南方车队
            waitCarQueue[SOUTH].size++;
            waitCarQueue[SOUTH].cars[waitCarQueue[SOUTH].tail++] = i
+ 1;

            break;
        case 'w':
            //西方
            car[i].id = i + 1;
            car[i].direction = 3;
            //加入西方车队
            waitCarQueue[WEST].size++;
            waitCarQueue[WEST].cars[waitCarQueue[WEST].tail++] = i +
1;

            break;
        default:

```

```

        break;
    }
}

//创建线程
for (int i = 0; i < strlen(argv[1]); i++)
{
    pthread_create(&carThread[i], NULL, carCross, (void *)&car
[i]);
}
//等待线程停止
for (int i = 0; i < strlen(argv[1]); i++)
{
    pthread_join(carThread[i], NULL);
}

//释放资源
pthread_mutex_destroy(&cross);
pthread_mutex_destroy(&deadlockLock);
pthread_cond_destroy(&deadlockCond);
for (int i = 0; i < 4; i++)
{
    pthread_mutex_destroy(&lock[i]);
    pthread_cond_destroy(&queue[i]);
    pthread_cond_destroy(&next[i]);
    pthread_mutex_destroy(&wait[i]);
}
}

void *carCross(void *car)
{
    struct carNode *Car = (struct carNode *)car;

    //如果这辆车不是等待列表中的第一辆车
    while (waitCarQueue[Car->direction].cars[waitCarQueue[Car->direction].head] != Car->id)
    {
        ;
    }
    //开到路口
    printf("car %d from %s arrives at crossing\n", Car->id, waitCar
Queue[Car->direction].direction);

    //开启路口锁

```



```

pthread_mutex_lock(&lock[Car->direction]);
//这个方向上有车在开, 等待信号量
while (isCar[Car->direction] == 1)
{
    pthread_cond_wait(&queue[Car->direction], &lock[Car->direction]);
}
//标记本车要开
isCar[Car->direction] = 1;
//标记本车等待
isWait[Car->direction] = 1;
//死锁检测
pthread_mutex_lock(&deadlockLock);
//发现死锁
if (isWait[NORTH] && isWait[EAST] && isWait[SOUTH] && isWait[WE
ST])
{
    switch (Car->direction)
    {
        case NORTH:
            //北方车让西方车先行驶
            printf("DEADLOCK: car jam detected, signalling West to g
o\n");
            pthread_cond_signal(&next[WEST]);
            break;
        case EAST:
            //东方车让北方车先行驶
            printf("DEADLOCK: car jam detected, signalling North to
go\n");
            pthread_cond_signal(&next[NORTH]);
            break;
        case SOUTH:
            //南方车让东方车先行驶
            printf("DEADLOCK: car jam detected, signalling East to g
o\n");
            pthread_cond_signal(&next[EAST]);
            break;
        case WEST:
            //西方车让北方车先行驶
            printf("DEADLOCK: car jam detected, signalling South to
go\n");
            pthread_cond_signal(&next[SOUTH]);
            break;
        default:

```

```

        break;
    }
}
pthread_mutex_unlock(&deadlockLock);
// 如果右边有车, 等待信号量
if (waitCarQueue[(Car->direction + 1) % 4].size != 0)
{
    pthread_cond_wait(&next[Car->direction], &lock[Car->direction]);
}
// 当前没有车在路口等待, 本车可以开了
isWait[Car->direction] = 0;
pthread_mutex_lock(&cross);
printf("car %d from %s leaving crossing\n", Car->id, waitCarQueue[Car->direction].direction);
pthread_mutex_unlock(&cross);
// 释放左边车辆通行信号
pthread_cond_signal(&next[(Car->direction + 3) % 4]);
pthread_mutex_unlock(&lock[Car->direction]);

// 将开走的车移除等待队列
pthread_mutex_lock(&Wait[Car->direction]);
waitCarQueue[Car->direction].head++;
waitCarQueue[Car->direction].size--;
isCar[Car->direction] = 0;
// 释放本车队下一辆车可以行驶的信号量
pthread_cond_signal(&queue[Car->direction]);
pthread_mutex_unlock(&Wait[Car->direction]);
pthread_exit(NULL);
}

```

2.1 题目

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、父进程的名字等；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，格式化输出（显示）内核模块输出的内容。

2.2 设计文档

2.2.3 内核模块

2.2.3.1 头文件

```
#include <linux/module.h>           //modules KERN_INFO
#include <linux/sched/signal.h>      //struct task_struct
```

头文件 linux/module.h 包含对模块的结构定义以及模块的版本控制，如 KERN_INFO。头文件 linux/sched/signal.h 包含必要的 INIT_TASK 宏。

2.2.3.2 主要函数

```
int init_module(void);
void cleanup_module(void);
```

函数 init_module 的作用是向内核注册模块提供的新功能。

函数 cleanup_module 的作用是注销所有模块注册的新功能。

2.2.3.3 init_module 说明

Linux 是一个多用户、多任务的系统，会产生很多的进程，每个进程会有不同的状态。

Linux 进程状态可以分为：可执行状态 TASK_RUNNING\可中断的睡眠状态

TASK_INTERRUPTIBLE\不可中断的睡眠状态 TASK_UNINTERRUPTIBLE\暂停状态

TASK_STOPPED\跟踪状态 TASK_TRACED\退出状态 TASK_DEAD\EXIT_ZOMBIE。

struct task_struct 中：

1. 成员 comm[TASKCOMMLEN]代表正在执行的名字，路径。
2. pid 代表进程的 id
3. state 代表进程状态,-1 代表没有运行，0 代表可以运行，大于 0 表示停止。
4. parent 代表父进程指针，children 代表孩子进程指针，sibling 代表兄弟进程指针。
5. exit_state 代表进程是否退出。

首先定义一个指向进程的指针 process，然后用这个指针遍历系统中所有的进程。Linux 系统中进程的组织数据结构是双向循环链表，其根为 init_task。因此，将 process 的初始值设为 init_task，通过 process=next_task(process)，将指针 process 修改为指向当前进程的下一个进程。通过判断 process 是否重新指向 init_task 决定循环是否结束。

对于系统中的每一个进程，首先通过在循环中 totalProcess++，统计系统中进程总数。读取结构体成员，在终端打印进程的基本信息，如

process->comm,process->pid,process->state,process->parent->pid,process->parent->comm。

首先判断进程是否终止。如果 process->exit_state == 0 为 true 代表进程没有终止，如果 process->exit_state == 0 为 false 代表进程已经终止。

一个进程没有终止，`state` 域能够取 5 个互为排斥的值。

状态	描述
TASK_RUNNING	1. 进程正在被执行 2. 进程就绪
TASK_INTERRUPTIBLE	进程因为等待一些条件而被阻塞
TASK_UNINTERRUPTIBLE	进程因为等待一些条件而被阻塞
TASK_STOPPED	进程被停止执行
TASK_TRACED	进程被 debugger 等进程监视

一个进程终止了，在 `exit_state` 域可以取 2 个值。

状态	描述
EXIT_ZOMBIE	进程被终止，但父进程未终止，进程成为僵尸进程
EXIT_DEAD	进程最终状态

用 `switch-case` 语句，判断进程状态，记录不同状态下的进程个数。

最后输出系统中每个进程的：名字、进程 `pid`、进程的状态、父进程的名字等；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。

在 `init_module` 中，先打印了一个“Process log:”的标志，为了用户态程序能够从日志中找到需要的进程统计信息。

2.2.4 用户程序

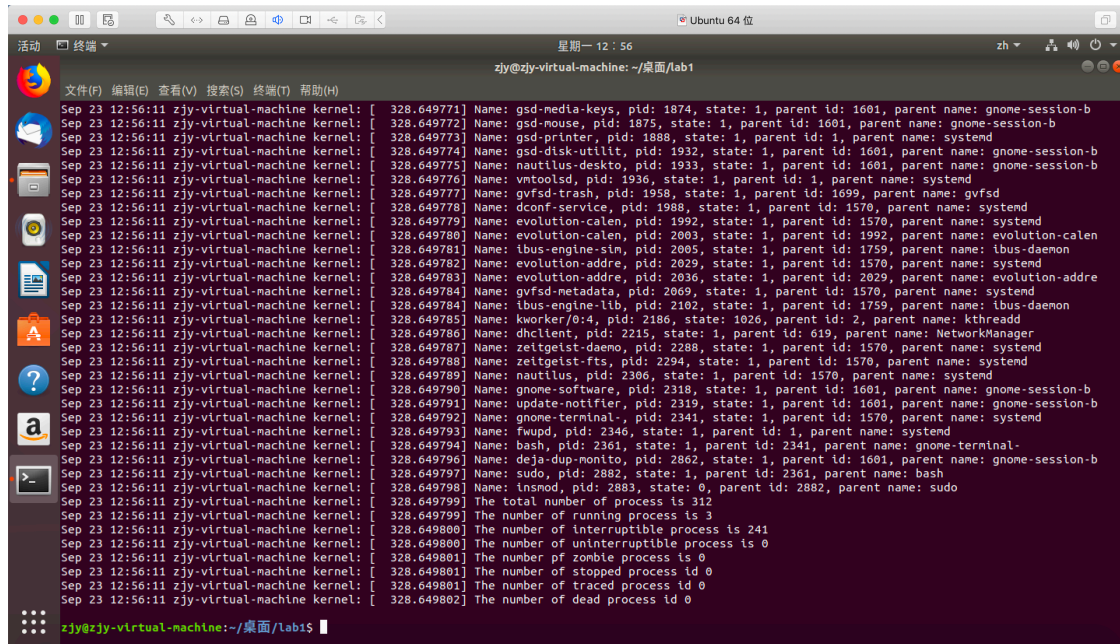
用户态下的程序是从 `/var/log/ kern.log (ubuntu)` 文件中读出内核模块输出的内容。首先，打开 `/var/log/ kern.log (ubuntu)` 文件。通过 `getline` 函数每次读取一行文件中的内容，通过 `find` 函数判断是否包含“Process log:”的提示信息，记录这个提示信息一共出现的次数 `count`。再次打开 `/var/log/ kern.log (ubuntu)` 文件，每次读取文件中的一行，统计“Process log:”提示信息在当前出现的次数 `flag`。当 `flag` 和 `count` 相等，说明下面的记录是需要打印的进程统计信息。于是向终端打印最后一次“Process log:”提示信息后出现的文件信息。

2.3 程序运行结果截图

1. `$make` 编译获得内核模块 `linuxKernel.ko`。
2. `$sudo insmod linuxKernel.ko` 加载内核模块。
3. `$g++ user.cpp` 编译获得可执行用户文件 `a.out`。
4. `$/a.out` 在当前目录下执行用户程序，得到运行进程统计结果。

```
zjy@zjy-virtual-machine: ~/桌面/lab1
[sudo] zjy 的密码:
rmmod: ERROR: Module linuxKernel is not currently loaded
zjy@zjy-virtual-machine: ~/桌面/lab1$ sudo insmod linuxKernel.ko
zjy@zjy-virtual-machine: ~/桌面/lab1$ ./a.out
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649494] Process log:
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649499] Name: systemd, pid: 1, state: 1, parent id: 0, parent name: swapper/0
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649500] Name: kthreadd, pid: 2, state: 1, parent id: 0, parent name: swapper/0
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649501] Name: rcu_gp, pid: 3, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649502] Name: rcu_par_gp, pid: 4, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649503] Name: kworker/0:0, pid: 5, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649506] Name: kworker/0:0H, pid: 6, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649507] Name: mm_percpu_wq, pid: 8, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649508] Name: ksoftirqd/0, pid: 9, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649508] Name: rcu_sched, pid: 10, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649509] Name: migration/0, pid: 11, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649510] Name: idle_inject/0, pid: 12, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649511] Name: kworker/0:1, pid: 13, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649512] Name: cpuhp/0, pid: 14, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649513] Name: kdevtmpfs, pid: 15, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649514] Name: netns, pid: 16, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649515] Name: rcu_tasks_kthre, pid: 17, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649516] Name: kauditd, pid: 18, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649517] Name: khungtaskd, pid: 19, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649518] Name: oom_reaper, pid: 20, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649518] Name: writeback, pid: 21, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649519] Name: kcompactd0, pid: 22, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649520] Name: ksmnd, pid: 23, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649521] Name: khugepaged, pid: 24, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649522] Name: crypto, pid: 25, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649523] Name: kintegrityd, pid: 26, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649524] Name: kblockd, pid: 27, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649525] Name: tpm_dev_wq, pid: 28, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649526] Name: ata_sff, pid: 29, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649527] Name: nd, pid: 30, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649528] Name: edac-poller, pid: 31, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649529] Name: devfreq_wq, pid: 32, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649530] Name: watchdogd, pid: 33, state: 1, parent id: 2, parent name: kthreadd
```

```
zjy@zjy-virtual-machine: ~/桌面/lab1
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649603] Name: scsi_eh_18, pid: 286, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649606] Name: scsi_eh_19, pid: 289, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649607] Name: scsi_tmf_19, pid: 290, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649608] Name: scsi_eh_20, pid: 291, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649609] Name: scsi_tmf_20, pid: 292, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649610] Name: scsi_eh_21, pid: 293, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649610] Name: scsi_tmf_21, pid: 294, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649611] Name: scsi_eh_22, pid: 295, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649612] Name: scsi_tmf_22, pid: 296, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649613] Name: scsi_eh_23, pid: 297, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649614] Name: scsi_tmf_23, pid: 298, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649615] Name: scsi_eh_24, pid: 299, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649616] Name: scsi_tmf_24, pid: 300, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649617] Name: scsi_eh_25, pid: 301, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649617] Name: scsi_tmf_25, pid: 302, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649618] Name: scsi_eh_26, pid: 303, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649619] Name: scsi_tmf_26, pid: 304, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649620] Name: scsi_eh_27, pid: 305, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649621] Name: scsi_tmf_27, pid: 306, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649622] Name: scsi_eh_28, pid: 307, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649622] Name: scsi_eh_29, pid: 308, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649623] Name: scsi_tmf_28, pid: 309, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649624] Name: scsi_eh_30, pid: 310, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649625] Name: scsi_tmf_29, pid: 311, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649626] Name: scsi_eh_31, pid: 312, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649627] Name: scsi_eh_31, pid: 313, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649629] Name: scsi_tmf_31, pid: 314, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649630] Name: scsi_eh_32, pid: 315, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649631] Name: scsi_tmf_32, pid: 316, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649632] Name: kworker/u256:28, pid: 340, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649633] Name: kworker/u256:29, pid: 341, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649635] Name: jbd2/sda1-8, pid: 368, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649635] Name: ext4-rsv-conver, pid: 369, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649637] Name: systemd-journal, pid: 414, state: 1, parent id: 1, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649637] Name: systemd-udev, pid: 427, state: 1, parent id: 1, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649638] Name: loop0, pid: 432, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649639] Name: loop1, pid: 433, state: 1, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649640] Name: loop2, pid: 435, state: 1, parent id: 2, parent name: kthreadd
```



```
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649771] Name: gsd-media-keys, pid: 1874, state: 1, parent id: 1601, parent name: gnome-session-b
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649772] Name: gsd-mouse, pid: 1875, state: 1, parent id: 1601, parent name: gnome-session-b
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649773] Name: gsd-printer, pid: 1888, state: 1, parent id: 1, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649774] Name: gsd-disk-utilit, pid: 1932, state: 1, parent id: 1601, parent name: gnome-session-b
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649775] Name: nautilus-deskto, pid: 1933, state: 1, parent id: 1601, parent name: gnome-session-b
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649776] Name: vntoolsd, pid: 1936, state: 1, parent id: 1, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649777] Name: gvfsd-trash, pid: 1958, state: 1, parent id: 1699, parent name: gvfsd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649778] Name: dconf-service, pid: 1988, state: 1, parent id: 1570, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649779] Name: evolution-calen, pid: 1992, state: 1, parent id: 1570, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649780] Name: evolution-calen, pid: 2003, state: 1, parent id: 1992, parent name: evolution-calen
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649781] Name: ibus-engine-stn, pid: 2005, state: 1, parent id: 1759, parent name: ibus-daemon
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649782] Name: evolution-adre, pid: 2029, state: 1, parent id: 1570, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649783] Name: evolution-adre, pid: 2036, state: 1, parent id: 2029, parent name: evolution-adre
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649784] Name: gvfsd-metadata, pid: 2069, state: 1, parent id: 1570, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649784] Name: ibus-engine-lib, pid: 2102, state: 1, parent id: 1759, parent name: ibus-daemon
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649785] Name: kworker/0:4, pid: 2186, state: 1026, parent id: 2, parent name: kthreadd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649786] Name: dhclient, pid: 2215, state: 1, parent id: 619, parent name: NetworkManager
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649787] Name: zeitgeist-daemo, pid: 2288, state: 1, parent id: 1570, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649788] Name: zeitgeist-fts, pid: 2294, state: 1, parent id: 1570, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649789] Name: nautilus, pid: 2306, state: 1, parent id: 1570, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649790] Name: gnome-software, pid: 2318, state: 1, parent id: 1601, parent name: gnome-session-b
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649791] Name: update-notifier, pid: 2319, state: 1, parent id: 1601, parent name: gnome-session-b
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649792] Name: gnome-terminal-, pid: 2341, state: 1, parent id: 1570, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649793] Name: fwupd, pid: 2346, state: 1, parent id: 1, parent name: systemd
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649794] Name: bash, pid: 2361, state: 1, parent id: 2341, parent name: gnome-terminal-
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649796] Name: deja-dup-monito, pid: 2802, state: 1, parent id: 1601, parent name: gnome-session-b
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649797] Name: sudo, pid: 2882, state: 1, parent id: 2361, parent name: bash
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649798] Name: insmod, pid: 2883, state: 0, parent id: 2882, parent name: sudo
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649799] The total number of process is 312
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649799] The number of running process is 3
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649800] The number of interruptible process is 241
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649800] The number of uninterruptible process is 0
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649801] The number of zombie process is 0
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649801] The number of stopped process is 0
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649801] The number of traced process is 0
Sep 23 12:56:11 zjy-virtual-machine kernel: [ 328.649802] The number of dead process is 0
zjy@zjy-virtual-machine: ~/桌面/lab1$
```

2.4 源程序

2.4.1 内核模块

linuxKernel.c

```
#include <linux/module.h> //modules KERN_INFO
#include <linux/sched/signal.h> //struct task_struct

/*initilize the module*/
int init_module(void)
{
    printk("Hello!\n"); //mark to start count

    int totalProcess = 0; //the total number of process
    /*the number of different state process*/
    int running = 0, interruptible = 0, uninterruptible = 0, zombie
= 0, stopped = 0, traced = 0, dead = 0;

    struct task_struct *process; //pointer pointing to process
    for(process=&init_task;(process=next_task(process))!=&init_tas
k;)//iterate over the process list
    {
        /*print the basic data*/
        printk(KERN_INFO"Name: %s, pid: %d, state: %d, parent id: %
d, parent name: %s",process->comm,process->pid,process->state,proce
ss->parent->pid,process->parent->comm);
```

```

        if(process->exit_state == 0)    //the process is not dead yet
        {
            switch(process->state)    //count the number of process in
different states
            {
                case TASK_RUNNING:
                    running++;
                    break;
                case TASK_INTERRUPTIBLE:
                    interruptible++;
                    break;
                case TASK_UNINTERRUPTIBLE:
                    uninterruptible++;
                    break;
                case TASK_STOPPED:
                    stopped++;
                    break;
                case TASK_TRACED:
                    traced++;
                    break;
                default:
                    break;
            }
        }
        else    //the process is dead or zombie
        {
            switch(process->exit_state)
            {
                case EXIT_ZOMBIE:
                    zombie++;
                    break;
                case EXIT_DEAD:
                    dead++;
                    break;
                default:
                    break;
            }
        }
        totalProcess++;
    }

    /*print the count result*/
    printk(KERN_INFO"The total number of process is %d\n", totalPro

```

```

cess);
    printk(KERN_INFO"The number of running process is %d\n", running);
    printk(KERN_INFO"The number of interruptible process is %d\n", interruptible);
    printk(KERN_INFO"The number of uninterruptible process is %d\n", uninterruptible);
    printk(KERN_INFO"The number pf zombie process is %d\n", zombie);
    printk(KERN_INFO"The number of stopped process id %d\n", stopped);
    printk(KERN_INFO"The number of traced process id %d\n", traced);
    printk(KERN_INFO"The number of dead process id %d\n", dead);

    return 0;
}

/*clean up the module*/
void cleanup_module(void)
{
    printk("Good-bye!\n");
}

MODULE_LICENSE("GPL");

```

2.4.2 用户程序

user.cpp

```

#include <iostream>
#include <fstream>
#include <string>

using namespace std;
#define MAXLENGTH 1000

int main()
{
    /*The program in user mode reads out the output of the kernel module

```



```

    */from the / var / log / kern. log (ubuntu) file
    */
    string file = "/var/log/kern.log";
    string mark = "Process log:";    //mark in the module

    int count = 0;    //count the number of hello occur

    char buffer[MAXLENGTH]; //store the one line log information
    string temp;

    fstream fs;
    fs.open(file, ios::in);

    while(!fs.eof())
    {
        fs.getline(buffer,MAXLENGTH,'\n');
        temp = buffer;
        if(temp.find(mark)!=string::npos)
        {
            count++;    //if hello occurs, count is added by 1
        }
    }
    fs.close();

    int flag = 0;
    fs.open(file, ios::in);
    while(!fs.eof())
    {
        fs.getline(buffer,MAXLENGTH,'\n');
        temp = buffer;
        if(temp.find(mark)!=string::npos)
        {
            flag++;
        }
        if(flag == count)    //if meet the last hello print the infor
        mation
        {
            cout<<buffer<<endl;
        }
    }
    fs.close();

    return 0;
}

```

三、讨论、心得（20 分）

1. 写好 helloworld.c 的程序后，编写 makefile 文件。根据《边干边学——Linux 内核指导》，KDIR 设置为 `KDIR=/usr/src/linux`

但是在编译时发现出现了错误。

```
zjy@zjy-virtual-machine:~/桌面/h$ make
make -C /usr/src/linux M=/home/zjy/桌面/h modules
make[1]: *** /usr/src/linux: 没有那个文件或目录。 停止。
Makefile:6: recipe for target 'default' failed
make: *** [default] Error 2
```

随后在上课听讲后，将 KDIR 修正为

```
KDIR = /lib/modules/$(shell uname -r)/build
```

```
make -C $(KDIR) M=$(PWD) modules
```

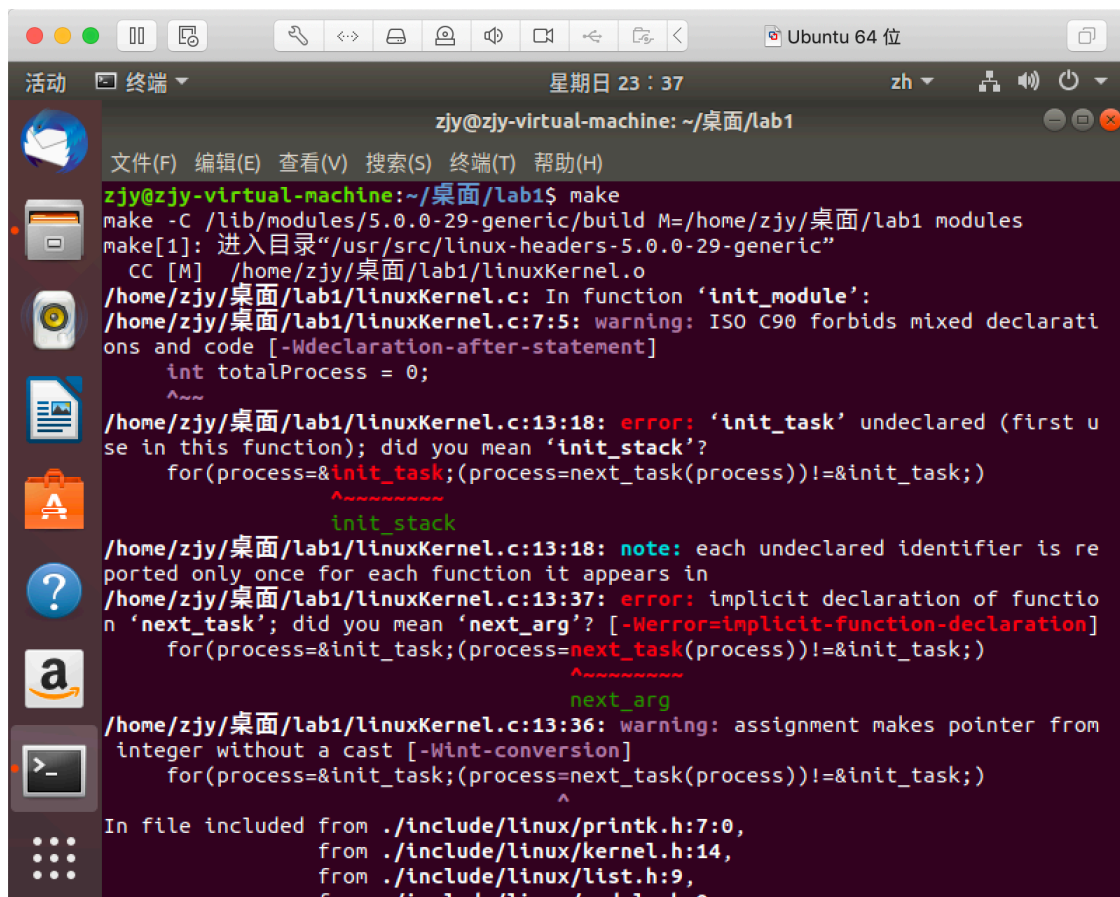
该命令中 -C 选项的作用是指将当前的工作目录转移到指定的目录（KDIR），程序到当前目录（pwd）查找模块源码，将其编译，生成.ko 文件。

2. 直接使用 insmod 执行内核模块失败。需要在命令前面加上 sudo 执行超级用户权限。

```
zjy@zjy-virtual-machine:~/桌面/h$ insmod h.ko
insmod: ERROR: could not insert module h.ko: Operation not permitted
zjy@zjy-virtual-machine:~/桌面/h$ sudo insmod h.ko
[sudo] zjy 的密码:
zjy@zjy-virtual-machine:~/桌面/h$
```

在 dmesg 后看到了 `[1004.596054] hello world` 的信息。

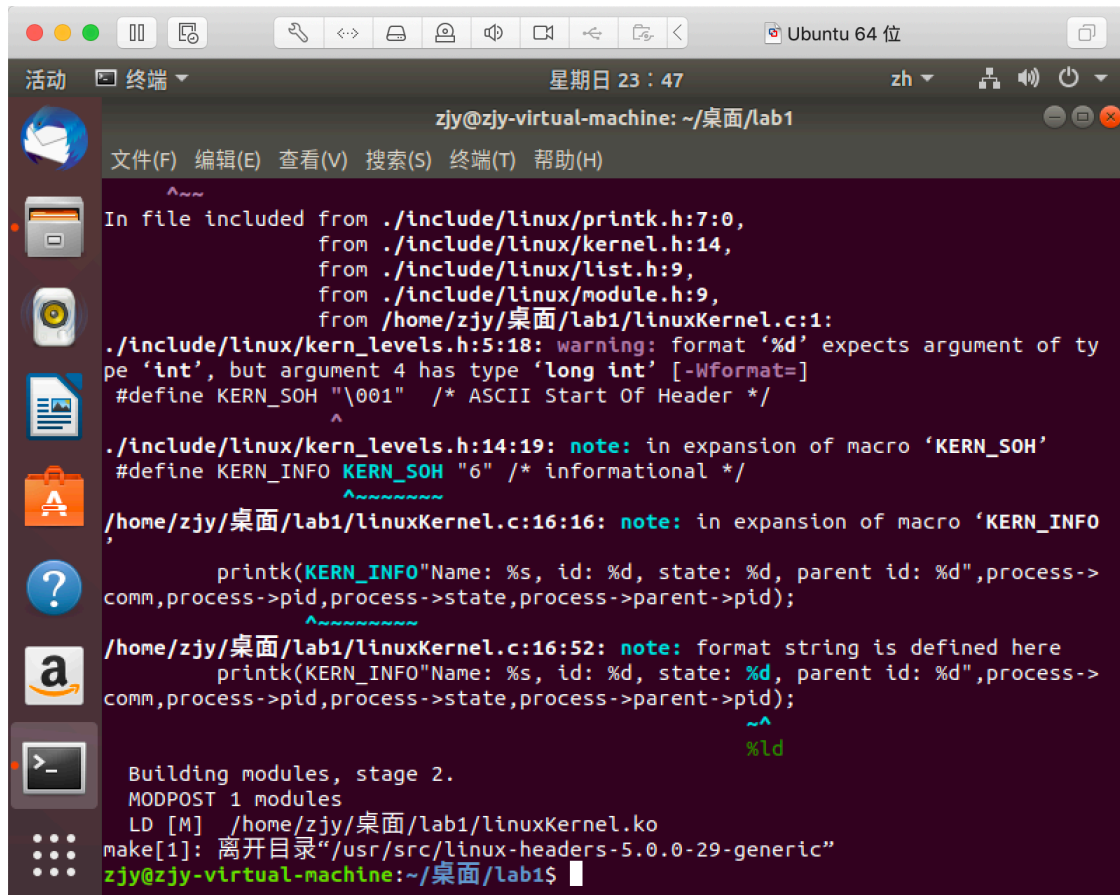
3. 在执行 makefile 时，会出现下面的错误。



```
zjy@zjy-virtual-machine: ~/桌面/lab1
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
zjy@zjy-virtual-machine:~/桌面/lab1$ make
make -C /lib/modules/5.0.0-29-generic/build M=/home/zjy/桌面/lab1 modules
make[1]: 进入目录"/usr/src/linux-headers-5.0.0-29-generic"
CC [M] /home/zjy/桌面/lab1/linuxKernel.o
/home/zjy/桌面/lab1/linuxKernel.c: In function 'init_module':
/home/zjy/桌面/lab1/linuxKernel.c:7:5: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
    int totalProcess = 0;
    ^
/home/zjy/桌面/lab1/linuxKernel.c:13:18: error: 'init_task' undeclared (first use in this function); did you mean 'init_stack'?
    for(process=&init_task;(process=next_task(process))!=&init_task;)
                   ^
/home/zjy/桌面/lab1/linuxKernel.c:13:18: note: each undeclared identifier is reported only once for each function it appears in
/home/zjy/桌面/lab1/linuxKernel.c:13:37: error: implicit declaration of function 'next_task'; did you mean 'next_arg'? [-Werror=implicit-function-declaration]
    for(process=&init_task;(process=next_task(process))!=&init_task;)
                                   ^
/home/zjy/桌面/lab1/linuxKernel.c:13:36: warning: assignment makes pointer from integer without a cast [-Wint-conversion]
    for(process=&init_task;(process=next_task(process))!=&init_task;)
                                   ^
In file included from ./include/linux/ printk.h:7:0,
                 from ./include/linux/kernel.h:14,
                 from ./include/linux/list.h:9,
                 from ./include/linux/module.h:8,
```

根据提示，`init_task`在使用前没有被定义。但 `init_task` 属于内核代码段，说明没有将正确的 Linux 源文件包含进去。查阅发现 4.11 以后，该方法都放在了 `include/linux/sched/signal.h` 中。修改后，编译成功。

```
zjy@zjy-virtual-machine:~/桌面/lab1$ cat /proc/version
Linux version 5.0.0-29-generic (build@lgw01-amd64-039) (gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #31~18.04.1-Ubuntu SMP Thu Sep 12 18:29:21 UTC 2019
```



The screenshot shows a terminal window titled "Ubuntu 64 位" with the command prompt "zjy@zjy-virtual-machine: ~/桌面/lab1". The terminal output displays several warnings and errors during the compilation of a kernel module. The warnings include a format string mismatch in `./include/linux/kern_levels.h:5:18` and a macro expansion issue in `./include/linux/kern_levels.h:14:19`. The error is a format string mismatch in `/home/zjy/桌面/lab1/linuxKernel.c:16:52`, where the format string `"Name: %s, id: %d, state: %d, parent id: %d"` expects an argument of type `'int'`, but the argument `process->parent->pid` has type `'long int'`. The terminal also shows the command `make[1]: 离开目录"/usr/src/linux-headers-5.0.0-29-generic"` and the prompt `zjy@zjy-virtual-machine:~/桌面/lab1$`.

```
In file included from ./include/linux/printk.h:7:0,
                  from ./include/linux/kernel.h:14,
                  from ./include/linux/list.h:9,
                  from ./include/linux/module.h:9,
                  from /home/zjy/桌面/lab1/linuxKernel.c:1:
./include/linux/kern_levels.h:5:18: warning: format '%d' expects argument of type 'int', but argument 4 has type 'long int' [-Wformat=]
#define KERN_SOH "\001" /* ASCII Start Of Header */
                  ^
./include/linux/kern_levels.h:14:19: note: in expansion of macro 'KERN_SOH'
#define KERN_INFO KERN_SOH "6" /* informational */
                  ^~~~~~
/home/zjy/桌面/lab1/linuxKernel.c:16:16: note: in expansion of macro 'KERN_INFO'
    printk(KERN_INFO"Name: %s, id: %d, state: %d, parent id: %d",process->
    ^~~~~~
comm,process->pid,process->state,process->parent->pid);
                  ^~~~~~
/home/zjy/桌面/lab1/linuxKernel.c:16:52: note: format string is defined here
    printk(KERN_INFO"Name: %s, id: %d, state: %d, parent id: %d",process->
    ^~~~~~
comm,process->pid,process->state,process->parent->pid);
                  ^~
                  %ld

Building modules, stage 2.
MODPOST 1 modules
LD [M] /home/zjy/桌面/lab1/linuxKernel.ko
make[1]: 离开目录"/usr/src/linux-headers-5.0.0-29-generic"
zjy@zjy-virtual-machine:~/桌面/lab1$
```

4. 一开始以为 helloworld.ko 模块加载到内核中就会马上在终端打印一个"hello world", 发现并没有, 后才意识到要有用户程序去调用这个内核模块, 才会执行这个模块中的代码, 在终端上打印"hello world".
5. 死锁检测要有单独的线程。
6. 编译线程函数最后要加上-lpthread。