# 同步互斥和Linux内核模块

# 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是如何实现模块机制的，掌握如何编写模块程序并进一步掌握内核模块的机理。

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。

# 实验内容

1. 下面的程序P1-1.c使用pthread线程库创建两个线程，两个线程都要访问共享变量counter，程序代码如下：

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

int counter=0; /\*两个线程都能访问的共享变量 \*/

void thread1(void \*arg);

void thread2(void \*arg);

int main(int argc, char \*argv[])

{

pthread\_t id1,id2;

/\*创建第1个线程\*/

pthread\_create(&id1,NULL,(void \*)thread1, NULL);

/\*创建第2个线程\*/

pthread\_create(&id2,NULL,(void \*)thread2, NULL);

pthread\_join(id1,NULL); /\*等待第1个线程结束\*/

pthread\_join(id2,NULL); /\*等待第2个线程结束\*/

/\*打印counter最后的值\*/

printf("最后的counter值为%d\n",counter);

exit(0);

}

void thread1(void \*arg) /\*第1个线程执行代码\*/

{

int i,val;

for(i=1;i<=5;i++){

val=++counter;

printf("第1个线程：第%d次循环，第1次引用counter=%d\n",i,counter); /\*LINE A\*/

usleep(300); /\*睡眠或挂起300毫秒钟\*/

printf("第1个线程：第%d次循环，第2次引用counter=%d\n",i,counter); /\*LINE B\*/

counter=val;

}

}

void thread2(void \*arg) /\*第2个线程执行代码\*/

{

int i,val;

for(i=1;i<=5;i++){

val=++counter;

usleep(100); /\*睡眠或挂起100毫秒钟\*/

printf("第2个线程：第%d次循环，counter=%d\n",i,counter);

counter=val;

}

}

用gcc编译程序：

gcc –o P1-1 P1-1.c –lpthread

按照源程序的逻辑，两个线程各循环5次，每次循环对counter变量加1，最后打印的结果应该是10。在你的系统中，编译并运行程序，观察运行结果，请回答以下问题：

1. 程序运行后，最后打印的counter是多少？

6

1. 第1个线程中同一次循环的LINE A行的printf和LINE B行的printf值是否一样？

不一样，一次为2，一次为3

1. 假设程序中LINE A行和LINE B行之间有很多代码，并多次引用counter的值，线程的运行结果会如何？

结果完全不能确定count的值，因为运行过程中将会不时地切换到进程2运行代码，从而改变count的值，因此每一次引用的count的值都是不确定的。

1. 请你多次调整两个线程中usleep函数的参数（即调整线程的睡眠时间），最后打印的counter值会出现什么样的变化？

|  |  |  |
| --- | --- | --- |
| 进程1参数 | 进程2参数 | Counter值 |
| 30 | 800 | 5 |
| 30 | 1000 | 5 |
| 300 | 10 | 6 |
| 1000 | 10 | 6 |

以上是一些改变参数后的输出结果，经过测试发现最后count的值可能为5和6。而counter最后的值取决于最后运行的进程。如果为进程1，则counter的值为6；否则 counter的值为5。

结果不是10，因为最后的count值只和val变量有关，不管在printf函数执行前有无抢占， count值有无短暂的变化，最后的count值都会被重新赋值成val,因此两个进程中val的值变化都是连续的，由于进程2后创建，具有更高的优先级，所以进程2先运行，val第一次赋值为1，进程1的val第一次赋值为2，所以最后进程1的val值一定为6，而进程2的val值最后一定为5，.

因此最后的count值取决于最后一次count是被哪个进程中的val赋值的，如果是进程1，就是6，如果是进程2，就是5.

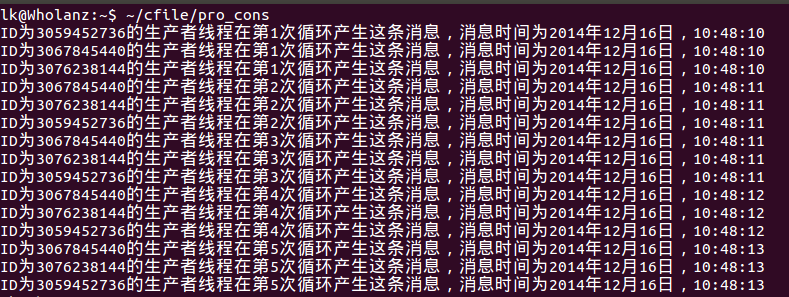
1. 请修改源程序，使用pthread的互斥锁机制，使得程序运行的结果正确，不管两个线程中的usleep函数的参数值多少，最后打印的counter值总是10。

修改后的程序源码如下：

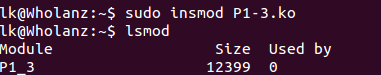
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <pthread.h>  int counter=0; /\*两个线程都能访问的共享变量 \*/  pthread\_mutex\_t mutex;  void thread1(void \*arg);  void thread2(void \*arg);  int main(int argc, char \*argv[])  {  pthread\_t id1,id2;  pthread\_create(&id1,NULL,(void \*)thread1, NULL);  /\*创建第1个线程\*/  /\*创建第2个线程\*/  pthread\_create(&id2,NULL,(void \*)thread2, NULL);  pthread\_mutex\_init(&mutex,NULL);  pthread\_join(id1,NULL); /\*等待第1个线程结束\*/  pthread\_join(id2,NULL); /\*等待第2个线程结束\*/  /\*打印counter最后的值\*/  pthread\_mutex\_destroy(&mutex);  printf("最后的counter值为%d\n",counter);  exit(0);  }  void thread1(void \*arg) /\*第1个线程执行代码\*/  {  int i,val;  for(i=1;i<=5;i++){  pthread\_mutex\_lock(&mutex);  val=++counter;    printf("第1个线程：第%d次循环，第1次引用counter=%d\n",i,counter); /\*LINE A\*/  usleep(10); /\*睡眠或挂起300毫秒钟\*/  printf("第1个线程：第%d次循环，第2次引用counter=%d\n",i,counter); /\*LINE B\*/    counter=val;  pthread\_mutex\_unlock(&mutex);  }  }  void thread2(void \*arg) /\*第2个线程执行代码\*/  {  int i,val;  for(i=1;i<=5;i++){  pthread\_mutex\_lock(&mutex);  val=++counter;  usleep(10); /\*睡眠或挂起100毫秒钟\*/  printf("第2个线程：第%d次循环，counter=%d\n",i,counter);  counter=val;  pthread\_mutex\_unlock(&mutex);  }  } |

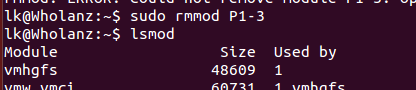
1. 通过这个程序例子,你得出什么结论?
2. 编写程序实现生产者-消费者经典进程同步问题。使用Linux的Pthread线程库，创建3个生产者线程和5个消费者线程。生产者线程计算当前的时间，把时间、第几次计算的序号（循环次数）和线程ID作为一个消息，把消息放入缓冲区，消费者线程从缓冲区读出一个消息并显示消息。缓冲区大小为8个，每个生产者线程生产5个消息，每个消费者线程消费3个消息，即生产15个消息和消费15个消息。

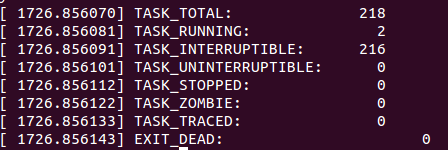
提示：3个生产者线程执行的代码可以是相同的（即执行同一函数），5个消费者线程也一样。



1. 编写一个Linux的内核模块，其功能是遍历进程，要求输出系统中：每个进程的名字、进程pid、进程的状态、父进程的名字；统计系统中进程个数，统计系统中TASK\_RUNNING、TASK\_INTERRUPTIBLE、TASK\_UNINTERRUPTIBLE、TASK\_ZOMBIE、TASK\_STOPPED等状态进程的个数。







# 实验指导

1. Linux线程创建和同步的编程参考资料：
2. 操作系统原理教材“Operating System Concepts”，第4章的4.3.1节，第6章的“Project”。
3. 短学期教材“Linux程序设计”，浙江大学出版社
4. Linux操作系统提供了pthread线程库，它是符合POSIX标准的函数库。线程控制方面的函数定义在pthread.h文件中、信号量控制方面的函数定义在semaphore.h文件中。

线程控制方面的函数有：pthread\_create、pthread\_join、pthread\_exit

信号量线程控制函数：sem\_init、sem\_wait、sem\_post、sem\_getvalue、sem\_destory

互斥锁机制函数：pthread\_mutex\_init、 pthread\_mutex\_lock、 pthread\_mutex\_unlock、 pthread\_mutex\_destroy

1. 使用gcc编译器编译C语言程序，gcc编译器的使用请参考“边干边学—Linux内核指导”教材第7章。编译第1、2题的程序时，在gcc中加入选项**-lpthread**，链接pthread线程库。
2. 第3题中，如何编写内核模块程序及编译、安装内核模块，可以参考“边干边学—Linux内核指导”教材第13章。
3. 第3题中，每个进程的进程名字、pid、进程状态、父进程的指针等在task-struct结构的字段中。在内核中使用printk函数打印有关变量的值。遍历进程可以使用next\_task宏，init\_task进程为0号进程。task-struct结构参阅“边干边学—Linux内核指导”教材11.2节；遍历进程方法可以参阅“边干边学—Linux内核指导”教材11.6节。

Makefile文件：

TARGET = helloworld //程序文件名

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

PWD = $(shell pwd)

obj-m += $(TARGET).o

default:

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

# 撰写实验报告的要求

1. 第2题需要提交程序实现的设计思路，必要的设计文档，源程序及注释，程序运行结果截图。
2. 按照实验报告模板格式撰写实验报告；
3. **源程序，程序代码的注释；**
4. **整个实验过程的截图；**
5. 实验过程中遇到的问题及解决方法等；
6. 心得体会。
7. 实验提交的内容包括实验报告+源代码（以文本格式单独存放在一个或几个文件中），所有的文件打包后再上传到“作业系统”中。