## 实验目的

* 加深对进程同步与互斥概念的认识；
* 掌握信号量的使用，并应用它解决生产者——消费者问题；
* 掌握信号量的实现原理。

## 实验内容

* 在 Ubuntu 下编写程序，用信号量解决生产者——消费者问题；
* 在 0.11 中实现信号量，用生产者—消费者程序检验之。

## 实验报告

|  |
| --- |
| * 在 pc.c 中去掉所有与信号量有关的代码，再运行程序，执行效果有变化吗？为什么会这样？ 实验的设计者在第一次编写生产者——消费者程序的时候，是这么做的：   Producer()  {  // 生产一个产品 item;  // 空闲缓存资源  P(Empty);  // 互斥信号量  P(Mutex);  // 将item放到空闲缓存中;  V(Mutex);  // 产品资源  V(Full);  }  Consumer()  {  P(Full);  P(Mutex);  //从缓存区取出一个赋值给item;  V(Mutex);  // 消费产品item;  V(Empty);  }   * 这样可行吗？如果可行，那么它和标准解法在执行效果上会有什么不同？如果不可行，那么它有什么问题使它不可行？ |
| 有变化，进程并发执行，又不能保护共享变量，会有不恰当的切换导致数据的一致性被破坏，产生脏数据。 |
| 这是标准写法，原意应该是吧PV mutex放在了外面，这就不对，会产生死锁，当产品满了时，生产者被阻塞在Empty上，消费者被阻塞在Mutex上，导致死锁。 |

## 实验步骤

1. unistd.h中创建信号量 sem\_t 及系统调用，注意结构体定义放在 \_\_LIBRARY\_\_ 的后面

|  |
| --- |
| 2020-05-14 20-05-52 的屏幕截图 |
| 2020-05-13 22-56-32 的屏幕截图 |
| 2020-05-13 22-51-38 的屏幕截图 |

1. Sys .h中系统调用表添加对应项

|  |
| --- |
| 2020-05-13 22-40-08 的屏幕截图  2020-05-13 21-07-36 的屏幕截图 |

1. 修改system\_call 中的系统调用数72改为76

|  |
| --- |
| 2020-05-13 22-39-51 的屏幕截图 |

1. 在 kernel 目录下sem .c中实现系统调用

注：”/r/n”和”/n”都能正常换行，效果相同

|  |
| --- |
| #include <unistd.h>  #include <errno.h>  #include <asm/segment.h>  #include <asm/system.h>  #include <linux/kernel.h>  //只用了三个同步量  static sem\_t sem\_array[3]={{{'\0'},0,NULL},{{'\0'},0,NULL},{{'\0'},0,NULL}};  sem\_t \*sys\_sem\_open(const char \*name, unsigned int value)  {  sem\_t\* ans=NULL;  char tmp\_name[64];  int i=0;  //复制最后的\0  for(i=0;tmp\_name[i]=get\_fs\_byte(name+i);i++);  for(i=0;i<3;i++)  {  if(!strcmp(tmp\_name,sem\_array[i].name))  {  ans=sem\_array+i;  #ifdef \_DEBUG\_  printk("%s,%s,%d\r\n",sem\_array[i].name,"reopen",sem\_array[i].value);  #endif  return ans;  }  }  for(i=0;i<3;i++)  {  if(sem\_array[i].name[0]=='\0')  {  strcpy(sem\_array[i].name,tmp\_name);  sem\_array[i].value=value;  sem\_array[i].block\_queue=NULL;  ans=sem\_array+i;  #ifdef \_DEBUG\_  printk("%s,%s,%d\r\n",sem\_array[i].name,"open",sem\_array[i].value);  #endif  return ans;  }  }  #ifdef \_DEBUG\_  printk("error open\r\n");  #endif  return ans;  }  int sys\_sem\_wait(sem\_t \*sem)  {  int ans=0;  cli();  if(sem<sem\_array||sem\_array+2<sem||sem->name[0]=='\0')  {  #ifdef \_DEBUG\_  printk("%p %p %p %s wait error\r\n",sem\_array,sem,sem\_array+2,sem->name);  #endif  sti();  return -1;  }  //挂载到队列直到sem->value>=0  while(sem->value<=0)  sleep\_on(&(sem->block\_queue));  --(sem->value);  sti();  #ifdef \_DEBUG\_  printk("%s,%s,%d\r\n",sem->name,"wait",sem->value);  #endif  return ans;  }  int sys\_sem\_post(sem\_t \*sem)  {  int ans=0;  cli();  if(sem<sem\_array||sem\_array+2<sem||sem->name[0]=='\0')  {  #ifdef \_DEBUG\_  printk("%p %p %p %s post error\r\n",sem\_array,sem,sem\_array+2,sem->name);  #endif  sti();  return -1;  }  ++(sem->value);  //不用while 只要等待队列还有唤醒让其继续运行就可  wake\_up(&(sem->block\_queue));  sti();  #ifdef \_DEBUG\_  printk("%s,%s,%d\r\n",sem->name,"post",sem->value);  #endif  return ans;  }  int sys\_sem\_unlink(const char \*name)  {  char tmp\_name[64];  int i=0;  for(i=0;tmp\_name[i]=get\_fs\_byte(name+i);i++);  for(i=0;i<3;i++)  if(!strcmp(name,sem\_array[i].name))  {  sem\_array[i].name[0]='\0';  #ifdef \_DEBUG\_  printk("%s,%s,%d\r\n",sem\_array[i].name,"unlink",sem\_array[i].value);  #endif  return 0;  }  #ifdef \_DEBUG\_  printk("unlink error\r\n");  #endif    return -1;  } |

1. 编辑makefile文件

|  |
| --- |
| 2020-05-13 22-31-39 的屏幕截图 |
| 2020-05-13 22-31-57 的屏幕截图 |

1. 挂载硬盘，替换头文件

|  |
| --- |
| 2020-05-13 23-02-44 的屏幕截图 |

1. 编写pc.c

|  |
| --- |
| #define \_\_LIBRARY\_\_  #include<stdio.h>  #include<unistd.h>  #include<stdlib.h>  #include<fcntl.h>  #define N 3  #define M 500  #define BUFSIZE 10  \_syscall2(sem\_t \*, sem\_open, const char\*,name,unsigned int,value);  \_syscall1(int,sem\_wait,sem\_t \*,sem);  \_syscall1(int,sem\_post,sem\_t \*,sem);  \_syscall1(int,sem\_unlink,const char \*,name);  int Producer();  int Consumer();  sem\_t \*empty=NULL,\*mutex=NULL,\*full=NULL;  char strbuf[32];  int fd=0;  int main()  {  int i=0;  pid\_t p;  freopen("out.txt","w",stdout);  empty=sem\_open("empty",BUFSIZE);  mutex=sem\_open("mutex",1);  full=sem\_open("full",0);    fd = open("./pc.data",O\_RDWR|O\_CREAT|O\_TRUNC,0777);  lseek(fd,M\*sizeof(int),SEEK\_SET);  write(fd,(char\*)&i,sizeof(int));  close(fd);  if(!fork())Producer();  for(i=0;i<N;i++)  {  if(!fork())  {  Consumer();  }  }  wait(&i);  return 0;  }  int Producer()  {  int data=1;  int loc=0;  do  {  sem\_wait(empty);  sem\_wait(mutex);  fd = open("./pc.data",O\_RDWR,0777);  lseek(fd,loc\*sizeof(int),SEEK\_SET);  write(fd,(char\*)&data,sizeof(int));  close(fd);  #ifdef \_MYDEBUG\_  sprintf(strbuf,"#locw%ddataw%d#\r\n",loc,data);  write(1,strbuf,strlen(strbuf));  #endif  loc++;  data++;  sem\_post(mutex);  sem\_post(full);  }while(data<=M);  exit(0);  }  int Consumer()  {  int data=0;  int loc=0;  int space=0;  while(data<M)  {  sem\_wait(full);  sem\_wait(mutex);    fd = open("./pc.data",O\_RDWR,0777);  lseek(fd,M\*sizeof(int),SEEK\_SET);  read(fd,(char\*)&loc,sizeof(int));  #ifdef \_MYDEBUG\_  sprintf(strbuf,"#locr%d#\r\n",loc);  write(1,strbuf,strlen(strbuf));  #endif  if(loc>=M)  {  sem\_post(mutex);  sem\_post(empty);  sem\_wait(empty);  sem\_post(full);  exit(0);  }  lseek(fd,loc\*sizeof(int),SEEK\_SET);  read(fd,(char\*)&data,sizeof(int));  sprintf(strbuf,"pid: %d\tdata:%d\r\n",getpid(),data);  write(1,strbuf,strlen(strbuf));    lseek(fd,loc\*sizeof(int),SEEK\_SET);  write(fd,(char\*)&space,sizeof(int));  loc++;  lseek(fd,M\*sizeof(int),SEEK\_SET);  write(fd,(char\*)&loc,sizeof(int));  #ifdef \_MYDEBUG\_  sprintf(strbuf,"#locw%d#\r\n",loc);  write(1,strbuf,strlen(strbuf));  #endif  close(fd);  sem\_post(mutex);  sem\_post(empty);  }  exit(0);  } |

1. 编译测试

|  |
| --- |
| 2020-05-14 00-00-17 的屏幕截图  2020-05-15 13-02-57 的屏幕截图2020-05-15 13-03-13 的屏幕截图 |