LOU1358487234 L32 2020-05-13 17:27:43 信号量的实现和应用

286 0

“操作系统原理与实践”实验报告

信号量的实现和应用

1. kernel/sem.c

实验数据

学习时间	0分钟
操作时间	0分钟
按键次数	0次
实验次数	1次
报告字数	9809字
是否完成	完成

评分

未评分

下一篇

篇

相关报告

- 操作系统原理与实践: 熟悉实验环境 实验报告
- 操作系统原理与实践: 熟悉实验环境 实验报告
- 操作系统原理与实践: 基于内核栈切换的进程切换 实验报告
- 操作系统原理与实践: 熟悉实验环境 实验报告
- 操作系统原理与实践: 信号量的实现和应用 实验报告

```
#define __LIBRARY__
#include <unistd.h>
#include <linux/kernel.h>
#include <asm/segment.h>
#include <asm/system.h>
#include <sys/types.h>
#include <linux/sched.h>
#define SEM_COUNT 32
sem_t semaphores[SEM_COUNT];
int enque(semq* sq, struct task_struct* task){
 if((sq->tail + 1) % QUE_NUM == sq->front) return -1;
 sq->task[sq->tail] = task;
 sq->tail ++;
 sq->tail %= QUE_NUM;
 return 1;
}
struct task_struct* deque(semq* sq){
 if(sq->tail == sq->front) return NULL;
 struct task_struct* ret = sq->task[sq->front];
 sq->front++;
 sq->front %= QUE_NUM;
 return ret;
}
void init_queue(semq* sq)
{
 sq->front = 0;
 sq->tail = 0;
 sq->enque = enque;
 sq->deque = deque;
}
/*打开信号量*/
sem_t* sys_sem_open(const char* name,unsigned int value)
{
 char tmp[16];
 char c;
 int i;
 for( i = 0; i<16; i++)
 {
     c = get_fs_byte(name+i);
     tmp[i] = c;
     if(c =='\0') break;
 }
 if(c >= 16) return NULL;
 for(i = 0;i< SEM_COUNT; i++)
 {
     if(semaphores[i].used != 0) continue;
 printk("%s saved in %d\n", tmp, i);
     strcpy(semaphores[i].name,tmp);
     semaphores[i].val = value;
 semaphores[i].used = 1;
     init_queue(&(semaphores[i].q));
     return &semaphores[i];
 }
 return NULL;
}
/*P原子操作*/
int sys_sem_wait(sem_t* sem)
{
 cli();
 sem->val--;
 if(sem->val < 0)
 {
     /*参见sleep_on*/
     current->state = TASK_UNINTERRUPTIBLE;
     sem->q.enque(&sem->q, current);
     schedule();
 }
 sti();
 return 0;
}
/*V原子操作*/
int sys_sem_post(sem_t* sem)
{
 cli();
 struct task_struct *p;
 sem->val++;
 if(sem->val <= 0)
 {
     p = sem->q.deque(&sem->q);
     if(p != NULL)
     {
         (*p).state = TASK_RUNNING;
     }
 }
 sti();
 return 0;
}
/*释放信号量*/
int sys_sem_unlink(const char *name)
{
 char tmp[16];
 char c;
 int i;
 for( i = 0; i<16; i++)
 {
     c = get_fs_byte(name+i);
     tmp[i] = c;
     if(c =='\0') break;
 }
 for(i = 0; i < SEM_COUNT; i++){
     if(semaphores[i].used == 1 && strcmp(semaphores[i].name, tmp) == 0){
     printk("Close %s\n", semaphores[i].name);
     semaphores[i].used = 0;
     break;
 }
 }
 return 0;
}
```

2. include/unistd.h

```
#define __NR_sem_open    72
#define __NR_sem_post    73
#define __NR_sem_wait    74
#define __NR_sem_unlink    75
#define QUE_NUM 10
typedef struct sem_queue{
 int front;
 int tail;
 struct task_struct* task[QUE_NUM];
 int (*enque)(struct sem_queue*, struct task_struct*);
 struct task_struct* (*deque)(struct sem_queue*);
}semq;
typedef struct{
 char name[16];
 int val;
 unsigned char used;
 semq q;
}sem_t;
#endif
```

3. pc.c

```
#define __LIBRARY__
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <stdio.h>
_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);
#define NUMBER 520 /*打出数字总数*/
#define CHILD 5 /*消费者进程数*/
#define BUFSIZE 10 /*缓冲区大小*/
sem_t   *empty, *full, *mutex;
int fno; /*文件描述符*/
int id = 0;
int main()
{
 int  i,j,k,p_id;
 int  data;
 pid_t p;
 int  buf_out = 0; /*从缓冲区读取位置*/
 int  buf_in = 0; /*写入缓冲区位置*/
 /*打开信号量*/
 if((mutex = sem_open("carmutex", 1)) == NULL)
 {
     perror("sem_open() error!\n");
     return -1;
 }
 if((empty = sem_open("carempty", 10)) == NULL)
 {
     perror("sem_open() error!\n");
     return -1;
 }
 if((full = sem_open("carfull", 0)) == NULL)
 {
     perror("sem_open() error!\n");
     return -1;
 }
 fno = open("buffer.dat", O_CREAT|O_RDWR|O_TRUNC, 0666);
 /* 将待读取位置存入buffer后,以便 子进程 之间通信 */
 lseek(fno, BUFSIZE*sizeof(int), SEEK_SET);
 write(fno, (char*)&buf_out, sizeof(int));
 /*生产者进程*/
 printf("Done1!\n");
 if((p=fork())==0)
 {
     for( i = 0 ; i < NUMBER; i++)
     {
         sem_wait(empty);
         sem_wait(mutex);
         /*写入一个字符*/
         lseek(fno, buf_in*sizeof(int), SEEK_SET);
         write(fno,(char*)&i,sizeof(int));
         buf_in = ( buf_in + 1)% BUFSIZE;

         sem_post(mutex);
         sem_post(full);
     }
     return 0;
 }else if(p < 0)
 {
     perror("Fail to fork!\n");
     return -1;
 }
 for( j = 0; j < CHILD ; j++ )
 {
     id++;
     if((p=fork())==0)
     {
         p_id = id;
         for( k = 0; k < NUMBER/CHILD; k++ )
         {
             sem_wait(full);
             sem_wait(mutex);
             /*获得读取位置*/
             lseek(fno,BUFSIZE*sizeof(int),SEEK_SET);
             read(fno,(char*)&buf_out,sizeof(int));
             /*读取数据*/
             lseek(fno,buf_out*sizeof(int),SEEK_SET);
             read(fno,(char*)&data,sizeof(int));
             /*写入读取位置*/
             buf_out = (buf_out + 1) % BUFSIZE;
             lseek(fno,BUFSIZE*sizeof(int),SEEK_SET);
             write(fno,(char*)&buf_out,sizeof(int));
             sem_post(mutex);
             sem_post(empty);
             /*消费资源*/
             printf("%d:  %d\n",p_id,data);
             fflush(stdout);
         }
        return 0;
     }else if(p<0)
     {
         perror("Fail to fork!\n");
         return -1;
     }
 }
 while(-1 != wait(NULL));
 /*释放信号量*/
 sem_unlink("carfull");
 sem_unlink("carempty");
 sem_unlink("carmutex");
 /*释放资源*/
 close(fno);
 return 0;
}
```

- 其实添加sem的时候应该先检测一下有没有同名的，这里省略了。
- 一开始pc.c里面设置消费者进程10个，结果输出一直是90%，浪费了大量时间在找原因上，后来突然想起来我等待队列设的长度就是10，循环队列有一个不能用...
- 实际调用可以是：

```
gcc -o pc pc.c
pc > 1.log
```

拿到ubuntu上看1.log即可。

```
5:  0
5:  1
5:  2
5:  3
5:  4
5:  5
4:  6
3:  7
2:  8
1:  9
4:  10
4:  11
4:  12
4:  13
4:  14
4:  15
3:  16
2:  17
1:  18
5:  19
4:  20
4:  21
4:  22
4:  23
4:  24
4:  25
3:  26
2:  27
1:  28
5:  29
4:  30
4:  31
4:  32
4:  33
4:  34
4:  35
3:  36
1:  37
2:  38
5:  39
3:  40
3:  41
3:  42
3:  43
3:  44
3:  45
1:  46
4:  47
2:  48
5:  49
1:  50
1:  51
1:  52
1:  53
1:  54
1:  55
4:  56
3:  57
2:  58
5:  59
4:  60
4:  61
4:  62
4:  63
4:  64
4:  65
3:  66
2:  67
1:  68
5:  69
4:  70
4:  71
4:  72
4:  73
4:  74
4:  75
3:  76
2:  77
5:  78
1:  79
4:  80
4:  81
4:  82
```