183975 L91 2018-02-14 18:25:08 信号量的实现和应用

680 0

“操作系统原理与实践”实验报告

信号量的实现和应用

一.添加信号量系统调用

1.修改unistd.h

```
/* add */
#include <sys/sem.h>
/* end */
```

```
/* add */
#define __NR_sem_open 72
#define __NR_sem_wait 73
#define __NR_sem_post 74
#define __NR_sem_unlink 75
/* end */
```

```
/* add */
sem_t *sem_open(const char *name, unsigned int value);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
int sem_unlink(const char *name);
/* end */
```

2.新建sem.h

```
/* add by jlb */
#ifndef _SEM_H
#define _SEM_H

/* include task_struct */
#include <linux/sched.h>

#define NAME_SIZE 10
/* add */
typedef struct { char name[NAME_SIZE]; unsigned int value; struct task_struct
*wait; char used; } sem_t;
/* end */

#endif
```

3.修改system_call.s

```
/* mod by jlb */
nr_system_calls = 76
```

4.修改sys.h

```
/* add */
extern int sys_sem_open();
extern int sys_sem_wait();
extern int sys_sem_post();
extern int sys_sem_unlink();
/* end */
```

实验数据

学习时间	707分钟
操作时间	539分钟
按键次数	12784次
实验次数	6次
报告字数	8528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

```
fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,
sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,
sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,
sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,
sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,
sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,
sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,
sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,
sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,
sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,
sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,
sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,
sys_setreuid, sys_setregid, sys_sem_open, sys_sem_wait, sys_sem_post,
sys_sem_unlink };
```

5.新增sem.c

```
/* add by jlb */
#include <sys/sem.h>
#include <errno.h>
/* include get_fs_byte() */
#include <asm/segment.h>
/* include NULL */
#include <unistd.h>
/* include cli() sti() */
#include <asm/system.h>
/* include prknik() */
#include <linux/kernel.h>

#define KSEM_SIZE 5

sem_t ksem[KSEM_SIZE] = {{{'0', }, 0, NULL, 0}, {{'\0', }, 0, NULL, 0}, {{'0',
}, 0, NULL, 0}, {{'0', }, 0, NULL, 0}, {{'0', }, 0, NULL, 0}};

char buf[NAME_SIZE];

int str_equ(const char * sa,const char * sb)
{
    int i;
    for (i=0; sa[i] && sb[i]; i++)
        if (sa[i] != sb[i])
            return 0;
    return 1;

}

void str_cpy(char * dest,const char * src)
{
    int i;
    for (i=0; dest[i] = src[i]; i++) ;
}

sem_t * sys_sem_open(const char *name, unsigned int value)
{
    int i;
    /* get name */
    for (i=0; buf[i] = get_fs_byte(name+i); i++) ;

    /* return old sem */
    for (i=0; i<KSEM_SIZE; i++)
        if (ksem[i].used) {
            if (str_equ(ksem[i].name, buf)) {
                /* debug */
                printk("old: buf: %s\n", buf);
                printk("old: %s, %d, %d, %d\n", ksem[i].name, ksem[i].value,
ksem[i].wait, ksem[i].used);
                return &ksem[i];
            }
        }

    /* create new sem */
    for (i=0; i<KSEM_SIZE; i++)
        if (!ksem[i].used) {
            str_cpy(ksem[i].name, buf);
            ksem[i].value = value;
            ksem[i].wait = NULL;
            ksem[i].used = 1;
            /* debug */
            printk("new: %s\n", buf);
            printk("new: %s, %d, %d, %d\n", ksem[i].name, ksem[i].value,
ksem[i].wait, ksem[i].used);
            return &ksem[i];
        }

    /* no sem */
    return NULL;
}

int sys_sem_wait(sem_t *sem)
{
    cli();
    while (!sem->value)
        sleep_on(&sem->wait);
    sem->value--;
    sti();
    return 0;
}

int sys_sem_post(sem_t *sem)
{
    cli();
    if (!sem->value)
        wake_up(&sem->wait);
    sem->value++;
    sti();
    return 0;
}

int sys_sem_unlink(const char *name)
{
    int i;
    /* get name */
    for (i=0; buf[i] = get_fs_byte(name+i); i++) ;

    /* try to delete sem */
    for (i=0; i<KSEM_SIZE; i++)
        if (ksem[i].used)
            if (str_equ(ksem[i].name, buf)) {
                ksem[i].used = 0;
                return 0;
            }
    return -ERROR;
}
```

6.修改makefile

```
# mod by jlb
OBJS  = sched.o system_call.o traps.o asm.o fork.o \
    panic.o printk.o vsprintf.o sys.o exit.o \
    signal.o mktime.o sem.o
```

```
# mod by jlb
sem.s sem.o : sem.c ../include/sys/sem.h ../include/errno.h
../include/asm/segment.h \
  ../include/unistd.h ../include/linux/sched.h ../include/asm/system.h
../include/linux/kernel.h
```

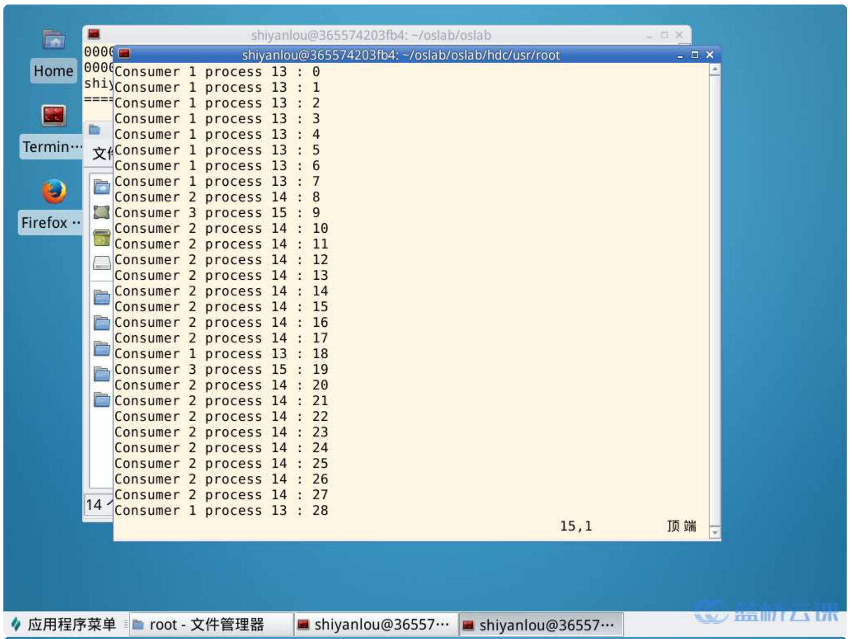
二.在文件系统中修改include文件中的头文件

如上所示，只须修改unistd.h和新增sem.h

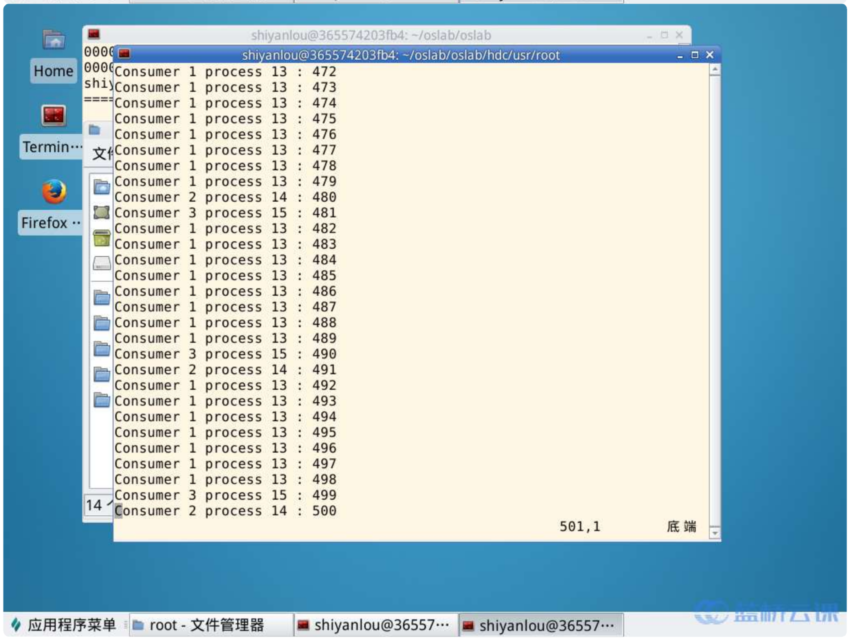
三.编写pc.c

```
#define __LIBRARY__
#include <unistd.h>
/* include some constant */
#include <fcntl.h>
/* include printf() fflush() */
#include <stdio.h>
#include <sys/sem.h>

_syscall0(int, fork)
_syscall0(int, getpid)
_syscall3(int, write, int, fildes, const char *, buf, off_t, count)
_syscall3(int, read, int, fildes, char *, buf, off_t, count)
_syscall3(int, lseek, int, fildes, off_t, offset, int, origin)
_syscall0(int, sync);

_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 BUF_SIZE 10

#define FILE_BUF_SIZE 10

int main(void)
{
    int pid;
    int fd_p, fd_c;
    /* count used by producer 1 */
    unsigned short count = 0;
    /* used by all consumers */
    unsigned short tmp = 0;
    unsigned char buf[BUF_SIZE] = {0};
    /* empty = 10, full = 0, mutex = 1. above 1 is unlocked, 0 is locked */
    char name_empty[10] = "empty";
    char name_full[10] = "full";
    char name_mutex[10] = "mutex";
    sem_t * empty, * full, * mutex;

    /* file, read and write */
    fd_p = open("./f", O_CREAT|O_TRUNC|O_RDWR, 0666);
    fd_c = open("./f", O_CREAT|O_TRUNC|O_RDWR, 0666);

    /* close old semaphore */
    sem_unlink(name_empty);
    sem_unlink(name_full);
    sem_unlink(name_mutex);

    /* open new semaphore */
    empty = sem_open(name_empty, FILE_BUF_SIZE);
    full = sem_open(name_full, 0);
    mutex = sem_open(name_mutex, 1);

    /* consumer 1 */
    if (!fork()) {
        pid = getpid();
        while (1) {
            sem_wait(full);
            sem_wait(mutex);

            read(fd_c, buf, sizeof(unsigned short));
            /* lower | higher */
            tmp = (unsigned short)buf[0] | ((unsigned short)buf[1] << 8);
            /* debug
            printf("%d, %d\n", (unsigned short)buf[0], (unsigned short)buf[1]
<< 8);
            */
            printf("Consumer 1 process %d : %d\n", pid, tmp);
            fflush(stdout);
            /* set file pos is 0 */
            if (tmp % FILE_BUF_SIZE == 9)
                lseek(fd_c, 0, SEEK_SET);

            sem_post(mutex);
            sem_post(empty);
        }
        return 0;
    }

    /* consumer 2 */
    if (!fork()) {
        pid = getpid();
        while (1) {
            sem_wait(full);
            sem_wait(mutex);

            read(fd_c, buf, sizeof(unsigned short));
            /* lower | higher */
            tmp = (unsigned short)buf[0] | ((unsigned short)buf[1] << 8);
            /* debug
            printf("%d, %d\n", (unsigned short)buf[0], (unsigned short)buf[1]
<< 8);
            */
            printf("Consumer 2 process %d : %d\n", pid, tmp);
            fflush(stdout);
            /* set file pos is 0 */
            if (tmp % FILE_BUF_SIZE == 9)
                lseek(fd_c, 0, SEEK_SET);

            sem_post(mutex);
            sem_post(empty);
        }
        return 0;
    }

    /* consumer 3 */
    if (!fork()) {
        pid = getpid();
        while (1) {
            sem_wait(full);
            sem_wait(mutex);

            read(fd_c, buf, sizeof(unsigned short));
            /* lower | higher */
            tmp = (unsigned short)buf[0] | ((unsigned short)buf[1] << 8);
            /* debug
            printf("%d, %d\n", (unsigned short)buf[0], (unsigned short)buf[1]
<< 8);
            */
            printf("Consumer 3 process %d : %d\n", pid, tmp);
            fflush(stdout);
            /* set file pos is 0 */
            if (tmp % FILE_BUF_SIZE == 9)
                lseek(fd_c, 0, SEEK_SET);

            sem_post(mutex);
            sem_post(empty);
        }
        return 0;
    }

    /* producer 1 */
    pid = getpid();
    while (count <= 500) {
        sem_wait(empty);
        sem_wait(mutex);

        /* get lower */
        buf[0] = (unsigned char)count;
        /* get higher */
        buf[1] = (unsigned char) (count >> 8);
        write(fd_p, buf, sizeof(unsigned short));
        sync();
        /* debug
        printf("%d, %d\n", buf[0], buf[1]);
        printf("Producer 1 process %d : %d\n", pid, count);
        fflush(stdout);
        */
        /* set file pos is 0 */
        if (count % FILE_BUF_SIZE == 9)
            lseek(fd_p, 0, SEEK_SET);
        count++;

        sem_post(mutex);
        sem_post(full);
    }

    return 0;
}
```

四.运行结果


Home
Termin…
Firefox ··
shiyanlou@365574203fb4: ~/oslab/oslab
shiyanlou@365574203fb4: ~/oslab/oslab/hdc/usr/root
应用程序菜单
root - 文件管理器
shiyanlou@36557…
shiyanlou@36557…

```
Consumer 1 process 13 : 0
Consumer 1 process 13 : 1
Consumer 1 process 13 : 2
Consumer 1 process 13 : 3
Consumer 1 process 13 : 4
Consumer 1 process 13 : 5
Consumer 1 process 13 : 6
Consumer 1 process 13 : 7
Consumer 2 process 14 : 8
Consumer 3 process 15 : 9
Consumer 2 process 14 : 10
Consumer 2 process 14 : 11
Consumer 2 process 14 : 12
Consumer 2 process 14 : 13
Consumer 2 process 14 : 14
Consumer 2 process 14 : 15
Consumer 2 process 14 : 16
Consumer 2 process 14 : 17
Consumer 1 process 13 : 18
Consumer 3 process 15 : 19
Consumer 2 process 14 : 20
Consumer 2 process 14 : 21
Consumer 2 process 14 : 22
Consumer 2 process 14 : 23
Consumer 2 process 14 : 24
Consumer 2 process 14 : 25
Consumer 2 process 14 : 26
Consumer 2 process 14 : 27
Consumer 1 process 13 : 28
                                                15,1          顶端
```


Home
Termin…
Firefox ··
shiyanlou@365574203fb4: ~/oslab/oslab
shiyanlou@365574203fb4: ~/oslab/oslab/hdc/usr/root
应用程序菜单
root - 文件管理器
shiyanlou@36557…
shiyanlou@36557…

```
Consumer 1 process 13 : 472
Consumer 1 process 13 : 473
Consumer 1 process 13 : 474
Consumer 1 process 13 : 475
Consumer 1 process 13 : 476
Consumer 1 process 13 : 477
Consumer 1 process 13 : 478
Consumer 1 process 13 : 479
Consumer 2 process 14 : 480
Consumer 3 process 15 : 481
Consumer 1 process 13 : 482
Consumer 1 process 13 : 483
Consumer 1 process 13 : 484
Consumer 1 process 13 : 485
Consumer 1 process 13 : 486
Consumer 1 process 13 : 487
Consumer 1 process 13 : 488
Consumer 1 process 13 : 489
Consumer 3 process 15 : 490
Consumer 2 process 14 : 491
Consumer 1 process 13 : 492
Consumer 1 process 13 : 493
Consumer 1 process 13 : 494
Consumer 1 process 13 : 495
Consumer 1 process 13 : 496
Consumer 1 process 13 : 497
Consumer 1 process 13 : 498
Consumer 3 process 15 : 499
Consumer 2 process 14 : 500
                                                501,1         底端
```

五.回答问题

完成实验后，在实验报告中回答如下问题：

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()
{
    P(Mutex);  //互斥信号量
    生产一个产品item;
    P(Empty);  //空闲缓存资源
    将item放到空闲缓存中;
    V(Full);  //产品资源
    V(Mutex);
}

Consumer()
{
    P(Mutex);
    P(Full);
    从缓存区取出一个赋值给item;
    V(Empty);
    消费产品item;
    V(Mutex);
}
```

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？

1.重复和乱，生产者和消费者没有秩序 2.首先Empty=10,Full=0,Mutex=1.不可行,可能造成死锁

4

B I

Markdown 语法

请输入想说的话

0 / 2000

发表评论

最新评论