



“操作系统原理与实践”实验报告

信号量的实现和应用

信号量的实现和应用

在linux 0.11上实现信号量

在实现的过程中将运用到系统调用的实验知识，通过此次实验也对前面做了一个比较系统的复习 ###在include/unistd.h文件中 定义如下数据结构，添加声明

```
*unistd.h (~/.oslab/oslab/linux-0.11/include) - gedit

File Edit View Search Tools Documents Help

time_t time(time_t * tloc);
time_t times(struct tms * tbuf);
int ulimit(int cmd, long limit);
mode_t umask(mode_t mask);
int umount(const char * specialfile);
int uname(struct utsname * name);
int unlink(const char * filename);
int ustat(dev_t dev, struct ustat * ubuf);
int utime(const char * filename, struct utimbuf * times);
pid_t waitpid(pid_t pid, int * wait_stat, int options);
pid_t wait(int * wait_stat);
int write(int fd, const char * buf, off_t count);
int dup2(int oldfd, int newfd);
int getpid(void);
pid_t getpgid(void);
pid_t setsid(void);

#define SEM_NAME_LEN 32
typedef struct sem_t{
    char name[SEM_NAME_LEN];
    unsigned int value;
    struct task_struct * s_wait; /*point to the process waiting the semaphore*/
    struct sem_t * next; /*linklist*/
}sem_t;

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);

#endif

C/C++/ObjC Header Tab Width: 8 Ln 263, Col 45 INS
```

```

shiyanolou@499d08a8e9b5: ~/oslab/oslab
oslab/linux-0.11/fs/file_dev.c
oslab/linux-0.11/fs/super.c
oslab/linux-0.11/fs/pipe.c
oslab/linux-0.11/fs/inode.c
oslab/linux-0.11/include - gedit
File Edit View Search Tools Documents Help
*unistd.h
#define _NR_ustat 62
#define _NR_dup2 63
#define _NR_getppid 64
#define _NR_getpgid 65
#define _NR_setsid 66
#define _NR_sigaction 67
#define _NR_sgetmask 68
#define _NR_ssetmask 69
#define _NR_setreuid 70
#define _NR_setregid 71
#define _NR_sem_open 72
#define _NR_sem_wait 73
#define _NR_sem_post 74
#define _NR_sem_unlink 75
#define _syscall0(type,name) \
type name(void) \
{ \
long res; \

```

应用程序菜单

蓝桥云课

在include/linux/sys.h文件中

添加声明，sys_call_table数组添加两个元素

```

shiyanolou@499d08a8e9b5: ~/oslab/oslab
oslab/linux-0.11/fs/pipe.c
oslab/linux-0.11/fs/inode.c
oslab/linux-0.11/fs/buffer.c
oslab/linux-0.11/fs/block_dev.c
oslab/linux-0.11/fs/bitmap.c
oslab/linux-0.11/fs/exec.c
oslab/linux-0.11/fs/char_dev.c
oslab/include/linux - gedit
File Edit View Search Tools Documents Help
*sys.h
extern int sys_sem_open();
extern int sys_sem_wait();
extern int sys_sem_post();
extern int sys_sem_unlink();
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 };

```

应用程序菜单

蓝桥云课

在kernel/system_call.s中

shiyanolou@499d08a8e9b5: ~/.oslab/oslab

oslab/linux-0.11/fs/buffer.c
oslab/linux-0.11/fs/block_dev.c
oslab/linux-0.11/fs/bitmap.c
oslab/linux-0.11/fs/exec.c

system_calls.s (~/.oslab/oslab/linux-0.11/kernel) - gedit

File Edit View Search Tools Documents Help

system_calls.s

```
priority = 8
signal = 12
sigaction = 16      # MUST be 16 (=len of sigaction)
blocked = (33*16)

# offsets within sigaction
sa_handler = 0
sa_mask = 4
sa_flags = 8
sa_restorer = 12

nr_system_calls = 76

/*
 * Ok, I get parallel printer interrupts while using the floppy for some
 * strange reason. Urgel. Now I just ignore them.
 */

.globl system_call,sys_fork,timer_interrupt,sys_execve
.globl hd_interrupt,floppy_interrupt,parallel_interrupt
.globl device_not_available,conprocessor_error
```

C Tab Width: 8 Ln 61, Col 21 INS

在此文件内实现4个山寨信号量函数。首先包含了必要的头文件，然后定义信号量队列的头指针。

```
#include <linux/kernel.h>
#include <asm/system.h>
#include <linux/sched.h>
#include <asm/segment.h>
#include <unistd.h>

sem_t *sem_head = &((sem_t *){"", 0, NULL, NULL}); /* head of the linklist */
```

```
/* copy sem's name from user stack to kernel stack */
static inline int str_u2k(const char *ustr, char *kstr, unsigned int length)
{
    char c;
    int i;

    for(i=0; (c=get_fs_byte(ustr++))!='\0' && i<length; i++)
        *(kstr+i)=c;
    *(kstr+i)='\0';

    return i;
}
```

下面进入正题，实现四个山寨的信号量函数

```

sem_t *sys_sem_open(const char *name, unsigned int value)
{
    sem_t *sem_cur, *sem_pre;
    char pname[SEM_NAME_LEN];

    str_u2k(name, pname, SEM_NAME_LEN);

    /*whether the sem existes in the linklist */
    for(sem_pre=sem_head, sem_cur=sem_head->next; sem_cur && strcmp(pname, sem_cur->name);
        sem_pre=sem_cur, sem_cur=sem_cur->next);

    /*if not exists, new a sem */
    if(!sem_cur)
    {
        printk("semaphore %s no found. created a new one. \n", pname);
        sem_cur = (sem_t *)malloc(sizeof(sem_t));
        strcpy(sem_cur->name, pname);
        sem_cur->value = value;
        sem_cur->next = NULL;
        sem_pre->next = sem_cur;
    }
    printk("pid %d opens semaphore %s(value %u) OK. \n", current->pid, pname, sem_cur->value);
    return sem_cur;
}

```

```

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

```

```

int sys_sem_post(sem_t *sem)
{
    cli();
    sem->value++;
    /* if still have process waiting for sem */
    if(sem->s_wait)
    {
        wake_up(&(sem->s_wait));
        sti();
        return 0;
    }
    sti();
    return -1;
}

```

```

int sys_sem_unlink(const char *name)
{
    sem_t *sem_cur, *sem_pre;
    char pname[SEM_NAME_LEN];
    int i;

    str_u2k(name, pname, SEM_NAME_LEN);

    for(sem_pre=sem_head, sem_cur=sem_head->next; sem_cur && strcmp(pname, sem_cur->name);
        sem_pre=sem_cur, sem_cur=sem_cur->next);

    /*if not found, return -1 */
    if(!sem_cur)
        return -1;

    /*if found, free it */
    sem_pre->next = sem_cur->next;
    free(sem_cur);
    printk("unlink semaphore %s OK. \n", pname);
    return 0;
}

```

18.2.4 : 之前大括号打错了，以至于fork了新的消费者而没参与到进程同步，现进行修正。此程序有时会出现死锁情况，原因不明。不过输出也是正确的，遇到死锁情况采取放置策略，重新启动系统运行程序即可

```

#define __LIBRARY__
#include <unistd.h>
#include <fcntl.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)

const char *FILENAME = "/usr/root/buffer_file";
const int NR_CONSUMERS = 5;          /* number of consumers */
const int NR_ITEMS = 50;             /* the maximum of item */
const int BUFFER_SIZE = 10;
sem_t *mutex, *full, *empty;
unsigned int item_pro, item_used;    /* the NO. of item which is produced or consumed just now */
int fi, fo;                          /* the I/O pointer*/

int main(int argc, char *argv[])
{
    char *filename;
    int pid;
    int i, flag = 1;

    filename = argc > 1 ? argv[1] : FILENAME;

    /* O_TRUNC means open the file in read-only or write-only, if exists, wipe the file*/
    fi = open(filename, O_CREAT| O_TRUNC| O_WRONLY, 0222); /* producer write-only */
    fo = open(filename, O_TRUNC| O_RDONLY, 0444);          /* consumer read-only */

    mutex = sem_open("MUTEX", 1);
    full = sem_open("FULL", 0);
    empty = sem_open("EMPTY", BUFFER_SIZE);

    item_pro = 0;

    if ((pid = fork())) /* father process is producer */
    {
        printf("pid %d:\tproducer created....\n", pid);
        fflush(stdout);

        while (item_pro < NR_ITEMS) /* while not finish produce the item */
        {
            sem_wait(empty);
            sem_wait(mutex);

            /* if the buffer is full, reset the pointer to the head of file */
            if(!(item_pro % BUFFER_SIZE))
                lseek(fi, 0, 0);

            write(fi, (char *) &item_pro, sizeof(item_pro)); /* produce */
            printf("pid %d:\tproduces item %d\n", pid, item_pro);
            fflush(stdout);
            item_pro++;

            sem_post(mutex);
            sem_post(full); /* wake up the consumer */
        }
    }
    else /* child process is consumer */
    {
        i = NR_CONSUMERS;
        while(i--)
        {
            if(!(pid=fork())) /* new the consumer */
            {
                pid = getpid();
                printf("pid %d:\tconsumer %d created....\n", pid, NR_CONSUMERS-i);
                fflush(stdout);

                while(1)
                {
                    sem_wait(full);
                    sem_wait(mutex);

                    /*when read() finish, return 0, reset the pointer to the head of file */

```



```

        if(!read(fo, (char *)&item_used, sizeof(item_used)))
        {
            lseek(fo, 0, 0);
            read(fo, (char *)&item_used, sizeof(item_used));
        }

        printf("pid %d:\tconsumer %d consumes item %d\n", pid, NR_CONSUMERS-i+1, item_used);
        fflush(stdout);

        sem_post(mutex);
        sem_post(empty);    /* wake up the producer */

        if(item_used == NR_ITEMS){    /* if all items have been consumed */
            flag = 0;
            goto OK;
        }
    }
}

}

OK:
while(flag);
sem_unlink("MUTEX");
sem_unlink("FULL");
sem_unlink("EMPTY");
close(fi);
close(fo);
return 0;
}

```

修改kernel/Makefile

运行程序

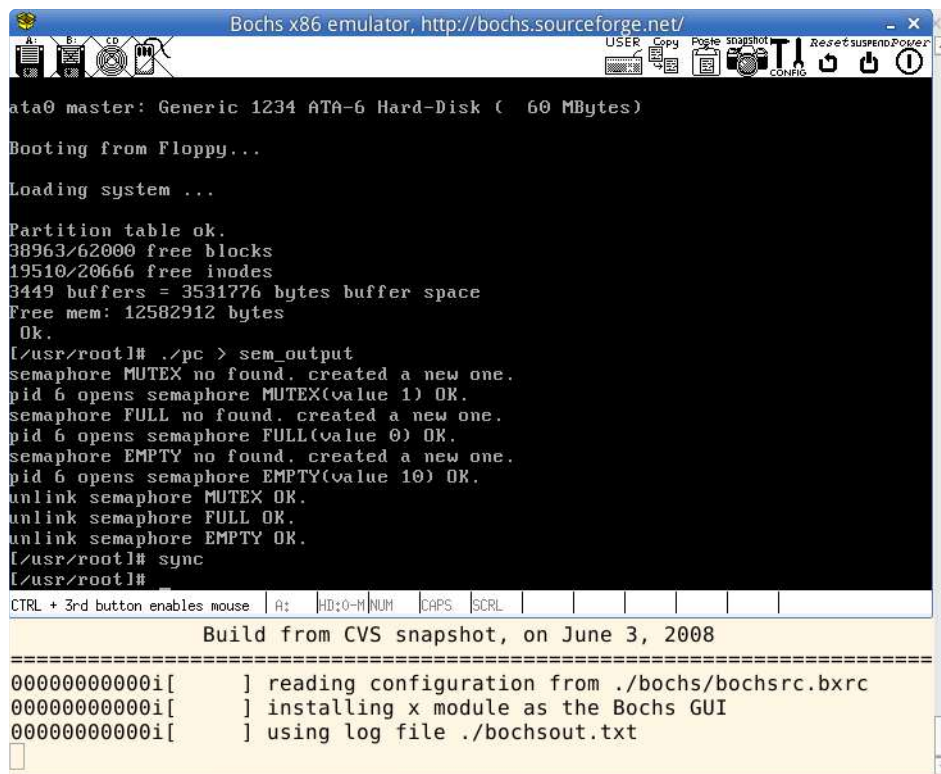
手动替换库文件

在oslab下输入`sudo ./mount-hdc`，将做过修改的库文件和`pc.c`拷贝进虚拟镜像之中，执行`sudo umount hdc`取消挂载

编译运行，查看结果

在linux 0.11下执行

```
gcc -o pc pc.c -Wall
./pc > sem_output
sync
```



```
ata0 master: Generic 1234 ATA-6 Hard-Disk ( 60 MBytes)
Booting from Floppy...
Loading system ...
Partition table ok.
38963/62000 free blocks
19510/20666 free inodes
3449 buffers = 3531776 bytes buffer space
Free mem: 12582912 bytes
Ok.
[/usr/root1# ./pc > sem_output
semaphore MUTEX no found. created a new one.
pid 6 opens semaphore MUTEX(value 1) OK.
semaphore FULL no found. created a new one.
pid 6 opens semaphore FULL(value 0) OK.
semaphore EMPTY no found. created a new one.
pid 6 opens semaphore EMPTY(value 10) OK.
unlink semaphore MUTEX OK.
unlink semaphore FULL OK.
unlink semaphore EMPTY OK.
[/usr/root1# sync
[/usr/root1#

CTRL + 3rd button enables mouse | A: | HD:0-M | NUM | CAPS | SCRL |
=====
Build from CVS snapshot, on June 3, 2008
=====
00000000000i[      ] reading configuration from ./bochs/bochsrc.bxrc
00000000000i[      ] installing x module as the Bochs GUI
00000000000i[      ] using log file ./bochsout.txt
```

应用程序菜单

退出前务必执行sync

在Ubuntu下执行sudo less ndc/usr/root/sem_output, 得到输出

```
shiyianlou@d0e9574fc318: ~/oslab/oslab
pid 7: producer created....
pid 7: produces item 0
pid 7: produces item 1
pid 7: produces item 2
pid 7: produces item 3
pid 7: produces item 4
pid 7: produces item 5
pid 7: produces item 6
pid 7: produces item 7
pid 7: produces item 8
pid 7: produces item 9
pid 12: consumer 5 created....
pid 12: consumer 6 consumes item 0
pid 12: consumer 6 consumes item 1
pid 12: consumer 6 consumes item 2
pid 12: consumer 6 consumes item 3
pid 12: consumer 6 consumes item 4
pid 12: consumer 6 consumes item 5
pid 12: consumer 6 consumes item 6
pid 12: consumer 6 consumes item 7
pid 12: consumer 6 consumes item 8
pid 12: consumer 6 consumes item 9
pid 11: consumer 4 created....
pid 10: consumer 3 created....
pid 9: consumer 2 created....
pid 8: consumer 1 created....
pid 7: produces item 10
pid 7: produces item 11
pid 7: produces item 12
:
```

应用程序菜单

```
shiyianlou@d0e9574fc318: ~/oslab/oslab
pid 7: produces item 12
pid 7: produces item 13
pid 7: produces item 14
pid 7: produces item 15
pid 7: produces item 16
pid 7: produces item 17
pid 7: produces item 18
pid 7: produces item 19
pid 8: consumer 2 consumes item 10
pid 8: consumer 2 consumes item 11
pid 8: consumer 2 consumes item 12
pid 8: consumer 2 consumes item 13
pid 8: consumer 2 consumes item 14
pid 8: consumer 2 consumes item 15
pid 8: consumer 2 consumes item 16
pid 8: consumer 2 consumes item 17
pid 8: consumer 2 consumes item 18
pid 8: consumer 2 consumes item 19
pid 7: produces item 20
pid 7: produces item 21
pid 7: produces item 22
pid 7: produces item 23
pid 7: produces item 24
pid 7: produces item 25
pid 7: produces item 26
pid 7: produces item 27
pid 7: produces item 28
pid 7: produces item 29
pid 12: consumer 6 consumes item 20
:
```

应用程序菜单

```
shiyianlou@d0e9574fc318: ~/oslab/oslab
pid 12: consumer 6 consumes item 20
pid 12: consumer 6 consumes item 21
pid 12: consumer 6 consumes item 22
pid 12: consumer 6 consumes item 23
pid 12: consumer 6 consumes item 24
pid 12: consumer 6 consumes item 25
pid 12: consumer 6 consumes item 26
pid 12: consumer 6 consumes item 27
pid 12: consumer 6 consumes item 28
pid 12: consumer 6 consumes item 29
pid 7: produces item 30
pid 7: produces item 31
pid 7: produces item 32
pid 7: produces item 33
pid 7: produces item 34
pid 7: produces item 35
pid 7: produces item 36
pid 7: produces item 37
pid 7: produces item 38
pid 7: produces item 39
pid 8: consumer 2 consumes item 30
pid 8: consumer 2 consumes item 31
pid 8: consumer 2 consumes item 32
pid 8: consumer 2 consumes item 33
pid 8: consumer 2 consumes item 34
pid 8: consumer 2 consumes item 35
pid 8: consumer 2 consumes item 36
pid 8: consumer 2 consumes item 37
pid 8: consumer 2 consumes item 38
:
```

应用程序菜单

```
shiyianlou@d0e9574fc318: ~/oslab/oslab
pid 8: consumer 2 consumes item 33
pid 8: consumer 2 consumes item 34
pid 8: consumer 2 consumes item 35
pid 8: consumer 2 consumes item 36
pid 8: consumer 2 consumes item 37
pid 8: consumer 2 consumes item 38
pid 8: consumer 2 consumes item 39
pid 7: produces item 40
pid 7: produces item 41
pid 7: produces item 42
pid 7: produces item 43
pid 7: produces item 44
pid 7: produces item 45
pid 7: produces item 46
pid 7: produces item 47
pid 7: produces item 48
pid 7: produces item 49
pid 12: consumer 6 consumes item 40
pid 12: consumer 6 consumes item 41
pid 12: consumer 6 consumes item 42
pid 12: consumer 6 consumes item 43
pid 12: consumer 6 consumes item 44
pid 12: consumer 6 consumes item 45
pid 12: consumer 6 consumes item 46
pid 12: consumer 6 consumes item 47
pid 12: consumer 6 consumes item 48
pid 12: consumer 6 consumes item 49
pid 7: produces item 50
pid 8: consumer 2 consumes item 50
(END)
```

应用程序菜单

与实验所要求的输出基本一致

实验报告问题

1.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？

执行效果变化很大，打印出的消费序列完全是乱的。因为没有信号量对临界区进行保护，加上进程调度是不确定的，导致生产者和消费者们的动作都无法做到互斥。也就是说,在临界区中完成一个动作之前，它们都有可能被另一个进程抢占，导致在缓冲区中的数据是混乱的。比如，消费者C正准备在缓冲区的起始处取出产品0时，就被生产者P抢占，而P已经完成了一轮的生产，也处于缓冲区的起始处，它往这里写入一个1。如果有轮到C执行。那么C取出的产品就是1，而不是0。

2.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. 假设Producer刚生产完一件商品，释放了Mutex，Mutex为1，此时缓存区满了，Empty为0；
- 2. 然后OS执行调度，若被Producer拿到CPU，它拿到Mutex，使Mutex为0，而Empty为0，Producer让出CPU，等待Consumer执行V(Empty)；
- 3. 而Consumer拿到CPU后，却要等待Producer执行V(Mutex)；
- 4. 两者相互持有对方需要的资源，造成死锁。

0

请 [登录](#) 后发表评论

最新评论



[LOU2979148015](#) **L19** 回复 [LOU2979148015](#) **L19**

直接在这回复也ok的

2019-05-21 22:51:48

回复



[LOU2979148015](#) **L19**

你好！我有一处不明，在pc.c中你好像只对item_pro变量进行了控制，而未对item_used进行控制。没想通是为什么。还望告知一下！谢谢你！qq2280343053

2019-05-21 22:51:26

回复