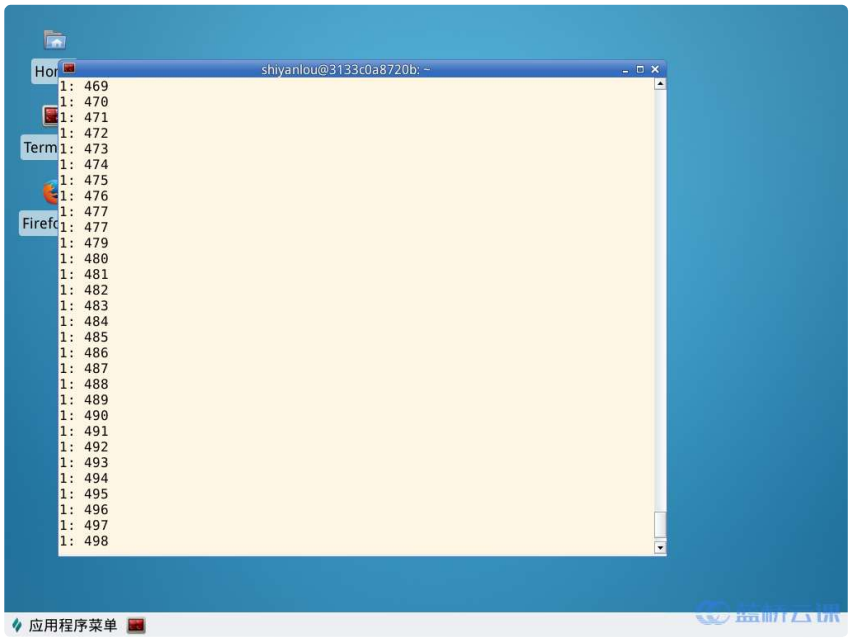
_refrain L73 2018-10-25 21:20:38 信号量的实现和应用

“操作系统原理与实践”实验报告

信号量的实现和应用



结果正确。相比上次，用了系统调用，而不是库函数，顺序立马就正常了。消费者使用记录的处理，通过将当前缓存的索引存在第11个数上实现，每次读取之前先读这个数得到索引，再读出这个位置上的数据。 输出文件：out.txt

在linux0.11上： unistd.h

实验数据

学习时间 1073分钟

操作时间 464分钟

按键次数 19520次

实验次数 5次

报告字数 12062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

```
#ifndef _UNISTD_H
#define _UNISTD_H

/* ok, this may be a joke, but I'm working on it */
#define _POSIX_VERSION 198808L

#define _POSIX_CHOWN_RESTRICTED    /* only root can do a chown (I think..) */
#define _POSIX_NO_TRUNC        /* no pathname truncation (but see in kernel) */
#define _POSIX_VDISABLE '\0'    /* character to disable things like ^C */
/*#define _POSIX_SAVED_IDS */    /* we'll get to this yet */
/*#define _POSIX_JOB_CONTROL */    /* we aren't there quite yet. Soon hopefully
*/

#define STDIN_FILENO    0
#define STDOUT_FILENO    1
#define STDERR_FILENO    2

#ifndef NULL
#define NULL    ((void *)0)
#endif

/* access */
#define F_OK    0
#define X_OK    1
#define W_OK    2
#define R_OK    4

/* lseek */
#define SEEK_SET    0
#define SEEK_CUR    1
#define SEEK_END    2

/* _SC stands for System Configuration. We don't use them much */
#define _SC_ARG_MAX        1
#define _SC_CHILD_MAX        2
#define _SC_CLOCKS_PER_SEC    3
#define _SC_NGROUPS_MAX        4
#define _SC_OPEN_MAX        5
#define _SC_JOB_CONTROL        6
#define _SC_SAVED_IDS        7
#define _SC_VERSION        8

/* more (possibly) configurable things - now pathnames */
#define _PC_LINK_MAX        1
#define _PC_MAX_CANON        2
#define _PC_MAX_INPUT        3
#define _PC_NAME_MAX        4
#define _PC_PATH_MAX        5
#define _PC_PIPE_BUF        6
#define _PC_NO_TRUNC        7
#define _PC_VDISABLE        8
#define _PC_CHOWN_RESTRICTED    9

#include <sys/stat.h>
#include <sys/times.h>
#include <sys/utsname.h>
#include <utime.h>

#ifdef __LIBRARY__

#define __NR_setup    0    /* used only by init, to get system going */
#define __NR_exit    1
#define __NR_fork    2
#define __NR_read    3
#define __NR_write    4
#define __NR_open    5
#define __NR_close    6
#define __NR_waitpid    7
#define __NR_creat    8
#define __NR_link    9
#define __NR_unlink    10
#define __NR_execve    11
#define __NR_chdir    12
#define __NR_time    13
#define __NR_mknod    14
#define __NR_chmod    15
#define __NR_chown    16
#define __NR_break    17
#define __NR_stat    18
#define __NR_lseek    19
#define __NR_getpid    20
#define __NR_mount    21
#define __NR_umount    22
```

```
#define __NR_setuid    23
#define __NR_getuid    24
#define __NR_stime    25
#define __NR_ptrace    26
#define __NR_alarm    27
#define __NR_fstat    28
#define __NR_pause    29
#define __NR_utime    30
#define __NR_stty    31
#define __NR_gtty    32
#define __NR_access    33
#define __NR_nice    34
#define __NR_ftime    35
#define __NR_sync    36
#define __NR_kill    37
#define __NR_rename    38
#define __NR_mkdir    39
#define __NR_rmdir    40
#define __NR_dup    41
#define __NR_pipe    42
#define __NR_times    43
#define __NR_prof    44
#define __NR_brk    45
#define __NR_setgid    46
#define __NR_getgid    47
#define __NR_signal    48
#define __NR_geteuid    49
#define __NR_getegid    50
#define __NR_acct    51
#define __NR_phys    52
#define __NR_lock    53
#define __NR_ioctl    54
#define __NR_fcntl    55
#define __NR_mpx    56
#define __NR_setpgid    57
#define __NR_ulimit    58
#define __NR_uname    59
#define __NR_umask    60
#define __NR_chroot    61
#define __NR_ustat    62
#define __NR_dup2    63
#define __NR_getppid    64
#define __NR_getpgrp    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; \
__asm__ volatile ("int $0x80" \
    : "=a" (__res) \
    : "0" (__NR_##name)); \
if (__res >= 0) \
    return (type) __res; \
errno = -__res; \
return -1; \
}

#define _syscall1(type,name,atype,a) \
type name(atype a) \
{ \
long __res; \
__asm__ volatile ("int $0x80" \
    : "=a" (__res) \
    : "0" (__NR_##name),"b" ((long)(a))); \
if (__res >= 0) \
    return (type) __res; \
errno = -__res; \
return -1; \
}

#define _syscall2(type,name,atype,a,btype,b) \
type name(atype a,btype b) \
{ \
```

```
long __res; \
__asm__ volatile ("int $0x80" \
    : "=a" (__res) \
    : "0" (__NR_##name),"b" ((long)(a)),"c" ((long)(b))); \
if (__res >= 0) \
    return (type) __res; \
errno = -__res; \
return -1; \
}

#define _syscall3(type,name,atype,a,btype,b,ctype,c) \
type name(atype a,btype b,ctype c) \
{ \
long __res; \
__asm__ volatile ("int $0x80" \
    : "=a" (__res) \
    : "0" (__NR_##name),"b" ((long)(a)),"c" ((long)(b)),"d" ((long)(c))); \
if (__res>=0) \
    return (type) __res; \
errno=-__res; \
return -1; \
}



#endif /* __LIBRARY__ */
struct sem_t{
    char name[20];
    int value;
    struct task_struct* queue;
};
typedef struct sem_t sem_t;

extern int errno;

int access(const char * filename, mode_t mode);
int acct(const char * filename);
int alarm(int sec);
int brk(void * end_data_segment);
void * sbrk(ptrdiff_t increment);
int chdir(const char * filename);
int chmod(const char * filename, mode_t mode);
int chown(const char * filename, uid_t owner, gid_t group);
int chroot(const char * filename);
int close(int fildes);
int creat(const char * filename, mode_t mode);
int dup(int fildes);
int execve(const char * filename, char ** argv, char ** envp);
int execv(const char * pathname, char ** argv);
int execvp(const char * file, char ** argv);
int execl(const char * pathname, char * arg0, ...);
int execlp(const char * file, char * arg0, ...);
int execle(const char * pathname, char * arg0, ...);
volatile void exit(int status);
volatile void _exit(int status);
int fcntl(int fildes, int cmd, ...);
int fork(void);
int getpid(void);
int getuid(void);
int geteuid(void);
int getgid(void);
int getegid(void);
int ioctl(int fildes, int cmd, ...);
int kill(pid_t pid, int signal);
int link(const char * filename1, const char * filename2);
int lseek(int fildes, off_t offset, int origin);
int mknod(const char * filename, mode_t mode, dev_t dev);
int mount(const char * specialfile, const char * dir, int rwflag);
int nice(int val);
int open(const char * filename, int flag, ...);
int pause(void);
int pipe(int * fildes);
int read(int fildes, char * buf, off_t count);
int setpgrp(void);
int setpgid(pid_t pid,pid_t pgid);
int setuid(uid_t uid);
int setgid(gid_t gid);
void (*signal(int sig, void (*fn)(int)))(int);
int stat(const char * filename, struct stat * stat_buf);
int fstat(int fildes, struct stat * stat_buf);
int stime(time_t * tptr);
int sync(void);
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 fildes, const char * buf, off_t count);
int dup2(int oldfd, int newfd);
int getppid(void);
pid_t getpgrp(void);
pid_t setsid(void);
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
```

sys.h

```
extern int sys_sigaction();
extern int sys_sgetmask();
extern int sys_ssetmask();
extern int sys_setreuid();
extern int sys_setregid();
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 };
```

system_call.s

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

nr_system_calls = 76

/*
```

sem.c

```
#include <unistd.h>
#include <errno.h>
#include <asm/segment.h>
#include <asm/system.h>
#define SEM_LIST_LENGTH 5

sem_t sem_list[5] = {{{'\0',},0,NULL},{{'\0',},0,NULL},{{'\0',},0,NULL},
{{'\0',},0,NULL},{{'\0',},0,NULL}};

sem_t * sys_sem_open(const char *name, unsigned int value){
    char nbuf[20];
    int i = 0;;
    for(i=0;nbuf[i]=get_fs_byte(name+i);i++);

    sem_t* result = NULL;//if fail, return NULL
    for(i=0;i<SEM_LIST_LENGTH;i++){
        //found
        if(!strcmp(sem_list[i].name,nbuf)){
            result = &sem_list[i];
            printk("sem %s found\n",result->name);
            return result;
        }
    }
    for(i=0;i<SEM_LIST_LENGTH;i++){
        //not found
        if(sem_list[i].name[0]=='\0'){
            strcpy(sem_list[i].name,nbuf);
            sem_list[i].value = value;
            sem_list[i].queue = NULL;
            result = &sem_list[i];
            printk("sem %s created, value = %d\n",result->name,result->value);
            return result;
        }
    }
    return result;
}

int sys_sem_wait(sem_t *sem){
    cli();
    //printk("in sem wait,sem value %d\n",sem->value);
    if(sem<sem_list||sem>sem_list+SEM_LIST_LENGTH){
        sti();
        printk("sem wait error\n");
        return -1;
    }
    while(sem->value<=0){
        sleep_on(&(sem->queue));
    }
    sem->value--;
    //printk("sem wait end\n");
    sti();
    return 0;
}

int sys_sem_post(sem_t *sem){
    cli();
    //printk("in sem post, sem value %d\n",sem->value);
    if(sem<sem_list||sem>sem_list+SEM_LIST_LENGTH){
        printk("sem wait error\n");
        sti();
        return -1;
    }
        sem->value++;
        wake_up(&(sem->queue));
        //printk("sem post end");
        sti();
        return 0;
}

int sys_sem_unlink(const char *name){
    char nbuf[20];
    int i = 0;;
    for(i=0;nbuf[i]=get_fs_byte(name+i);i++);
    for(i=0;i<SEM_LIST_LENGTH;i++){
        //found
        if(!strcmp(sem_list[i].name,nbuf)){
            printk("sem %s unlinked\n",sem_list[i].name);
            sem_list[i].name[0]='\0';
            sem_list[i].queue=NULL;
            return 0;
        }
    }
}
```

这里遇到了一些问题，就是task_struct传入的是指针的指针，因为这个问题搞了好久。还有就是wake_up的时候不要判断是不是NULL，这里在下面的调度已经做好了。这里画蛇添足造成进程一直等在这里。 pc.c

```c
#define __LIBRARY__
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<fcntl.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 int cNum = 5;
const int itemNum = 500;
const int bufSize = 10;

int main(){
    int data;
    int fd;
    sem_t *sem_empty,*sem_full,*sem_mutex;
    int buf_in=0,buf_out=0;
    int i,j,k;
    pid_t p;
    freopen("out.txt","w",stdout);
    printf("start\n");
    fflush(stdout);
    sem_unlink("empty");
    sem_unlink("full");
    sem_unlink("mutex");
    if((sem_empty=sem_open("empty",10))==NULL){
        perror("sem_empty error!\n");
        return -1;
    }
    if((sem_full=sem_open("full",0))==NULL){
        perror("sem_full error!\n");
        return -1;
    }
    if((sem_mutex=sem_open("mutex",1))==NULL){
        perror("sem_mutex error!\n");
        return -1;
    }
    fd = open("buffer.dat",O_RDWR|O_CREAT|O_TRUNC,777);

    lseek(fd,bufSize*sizeof(int),SEEK_SET);
    write(fd,&buf_out,sizeof(int));
    if(!(p=fork())){
        for(i=0;i<itemNum;i++){
            sem_wait(sem_empty);
            sem_wait(sem_mutex);
            lseek(fd,buf_in*sizeof(int),SEEK_SET);
            write(fd,(char*)&i,sizeof(int));
            buf_in = (buf_in+1)%bufSize;
            sem_post(sem_mutex);
            sem_post(sem_full);
        }
        return 0;
    }
    else if(p<0){
        perror("fork error!\n");
        return -1;
    }
    for(j=0;j<cNum;j++){
        if(!(p=fork())){
            for(k=0;k<itemNum/cNum;k++){
                sem_wait(sem_full);
                sem_wait(sem_mutex);
                lseek(fd,bufSize*sizeof(int),SEEK_SET);
                read(fd,(char*)&buf_out,sizeof(int));
                lseek(fd,buf_out*sizeof(int),SEEK_SET);
                read(fd,(char*)&data,sizeof(int));
                buf_out=(buf_out+1)%bufSize;
                lseek(fd,bufSize*sizeof(int),SEEK_SET);
                write(fd,(char*)&buf_out,sizeof(int));
                printf("%d:    %d\n",getpid(),data);
                fflush(stdout);
                sem_post(sem_mutex);
                sem_post(sem_empty);
            }
            return 0;
        }
        else if(p<0){
            perror("fork error!\n");
```

```
            return -1;
        }
    }
    sem_unlink("empty");
    sem_unlink("full");
    sem_unlink("mutex");
    close(fd);
    return 0;
}
```

这里就是老版本的gcc有点麻烦。数据段和代码段必须分开，而且不认识"//"这种注释。 终于搞好了！ #问题回答 不可行。 对于某生产者，当mutex=1,empty=0时，申请以后变为：mutex=0,empty=-1，阻塞。 同时对于某消费者，此时mutex=0，申请后变为mutex=-1，也发生阻塞。 所以会产生死锁。

1

Markdown 语法

请输入想说的话

0 / 2000

发表评论

最新评论