ab1214 L446 2019-07-08 16:21:47 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

信号量实验

1. 实验内容

- 在kernel 里实现open_sem();wait_sem();post_sem();unlink_sem();的系统调用。
- 写一个应用，实现生产者进程产生500个数字，和若干个消费者进程，利用上述系统调用完成依赖信号量的多进程同步。

2. 细节描述

- 新建信号量
  > 获得一个信号量结构体指针，并根据此指针进行后续信号量的操作；
- 进程的睡眠和唤醒
  > 在wait_sem中，信号量的值减一，并判断其值是否<0,如果是，将当前进程睡眠，并切换；而对应的在post_sem中，则需要将信号量+1，如果信号量<=0,则需要择一睡眠进程唤醒；
- 生产进程和消费者进程
  > 生产者进程和消费者进程依赖信号量同步自己的行为

3. 问题描述

- 父子进程间共享栈，代码段，数据段，但是当变量在进程中发生写操作时，linux会触发copy-on-write,所以进程间并*不能共享局部变量，全局变量，以及堆中的数据**，但出内存区的数据，其他如文件是可以共享的，所以多个消费者进程需要共享文件读指针偏移量（*offset*），需要将offset存入文件当中***，并在每次offset变化时，写入文件当中。
- 实验楼所有其他人都用了了0.11本身的sleep_on()，wake_up()函数来实现进程睡眠和唤醒，但是其唤醒方式会一次*唤醒所有**，故在*wait_sem*中需要使用*while*循环判断的方式，让所有进程再竞争一次。*shiyanlou*里所有其他人都是用此种方式实现的。但是我是按队列的方式，每次只唤醒一个进程***，所以使用if判断一次即可，
- 因为使用了记录睡眠队列的方式，所以我需要一个数组来保存睡眠的进程，自然的我在定义sem结构体的时候，里面**嵌套了一个*task_struct*结构体数组**，但**灾难**，来了，可能gcc对这种**复杂嵌套**无法处理，导致运行时结构体内的数据凌乱了，而且是调试了很久才发现的。浪费了大量的时间，
- 将pcb结构体数组移出sem的定义是无奈之举，但也促成了更好的方式，我定义了一个公共的pcb的全局数组，并加上了idle项表明其是否在使用，以及一个next_node项，来指向下一个pcb，方便一个信号量的所有等待进程，形成一个链式队列。
- linux + boch这种调试方式事倍功半，只能用printf来调试，效率极其低下，大量的时间用来发现和调试一个bug。

4. 代码

- sem.h

实验数据

学习时间	2495分钟
操作时间	600分钟
按键次数	21286次
实验次数	15次
报告字数	15849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 _SEM_H
#define _SEM_H
#define NR_SEM  32
#define NR_WAIT_NODE 20 //max size of sem-waiting queue
#ifndef NULL
#define NULL ((void *) 0)
#endif
#include <linux/sched.h>
typedef struct wait_node{
  int idle;
  struct task_struct* p_task_struct;
  struct wait_node * next_node;
} wait_node_t;
typedef struct  {
  int value;
   char name[20];
  wait_node_t* p_wait_queue;
} sem_t;
extern sem_t* sys_sem_open(const char* name,  int value);
extern int sys_sem_wait(sem_t* sem);
extern int sys_sem_post(sem_t* sem);
extern int sys_sem_unlink(const char* name);
extern void sem_init();
#endif
```

- unistd.h

```
#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__ */
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);
#endif
```

- sys.h

```
#include <sem.h>
extern int sys_setup();
extern int sys_exit();
extern int sys_fork();
extern int sys_read();
extern int sys_write();
extern int sys_open();
extern int sys_close();
extern int sys_waitpid();
extern int sys_creat();
extern int sys_link();
extern int sys_unlink();
extern int sys_execve();
extern int sys_chdir();
extern int sys_time();
extern int sys_mknod();
extern int sys_chmod();
extern int sys_chown();
extern int sys_break();
extern int sys_stat();
extern int sys_lseek();
extern int sys_getpid();
extern int sys_mount();
extern int sys_umount();
extern int sys_setuid();
extern int sys_getuid();
extern int sys_stime();
extern int sys_ptrace();
extern int sys_alarm();
extern int sys_fstat();
extern int sys_pause();
extern int sys_utime();
extern int sys_stty();
extern int sys_gtty();
extern int sys_access();
extern int sys_nice();
extern int sys_ftime();
extern int sys_sync();
extern int sys_kill();
extern int sys_rename();
extern int sys_mkdir();
extern int sys_rmdir();
extern int sys_dup();
extern int sys_pipe();
extern int sys_times();
extern int sys_prof();
extern int sys_brk();
extern int sys_setgid();
extern int sys_getgid();
extern int sys_signal();
extern int sys_geteuid();
extern int sys_getegid();
extern int sys_acct();
extern int sys_phys();
extern int sys_lock();
extern int sys_ioctl();
extern int sys_fcntl();
extern int sys_mpx();
extern int sys_setpgid();
extern int sys_ulimit();
extern int sys_uname();
extern int sys_umask();
extern int sys_chroot();
extern int sys_ustat();
extern int sys_dup2();
extern int sys_getppid();
extern int sys_getpgrp();
extern int sys_setsid();
extern int sys_sigaction();
extern int sys_sgetmask();
extern int sys_ssetmask();
extern int sys_setreuid();
extern int sys_setregid();
extern sem_t* 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_un
link };
```

- sem.c ```c /*
- linux/kernel/sem.c
-
- (C) 2019 Tsai Tsu Quen
- /

include <linux/sched.h>

include <linux/kernel.h>

include <linux/sem.h>

include <asm/system.h>

include <asm/segment.h>

sem_t sem_list[NR_SEM]; wait_node_t wait_node_list[NR_WAIT_NODE]; wait_node_t* _find_idle_wait_node() { int i = 0; for (i = 0; i < NR_WAIT_NODE; i++) {

```
if (wait_node_list[i].idle == 1)
{
    wait_node_list[i].idle = 0;
    wait_node_list[i].next_node = NULL;
    return (wait_node_t*)(wait_node_list + i);
}
```

} printk("no wait node\n"); return NULL; } int _str_compare(char* buf_1, char* buf_2) { int i = 0; while ((buf_1[i] == buf_2[i]) && (buf_1[i] != '\0')) {

```
i++;
```

} if (buf_1[i] == '\0' && buf_2[i] == '\0')

```
return 0;
```

else

```
return -1;
```

} int _str_copy(char* buf_s, char* buf_d) { int i = 0; do {

```
buf_d[i] = buf_s[i];
i++;
```

} while (buf_s[i] != '\0'); return 0; } sem_t* sys_sem_open(const char* name, int value) { int i = 0; int j = NR_SEM + 1; char name_buf[20]; while ((name_buf[i] = get_fs_byte(name + i)) != '\0')

```
i++;
```

for (i = 0; i < NR_SEM; i++) {

```
    if (*(sem_list[i].name) == '\0')
        j = i;
    else if (_str_compare(sem_list[i].name, name_buf) == 0)
        return (sem_t*)(sem_list + i);
```

} if (j < NR_SEM) {

```
    _str_copy(name_buf, sem_list[j].name);
    sem_list[j].value = value;
    sem_list[j].p_wait_queue = NULL;
    return (sem_t*)(sem_list + j);
```

} else

```
    return NULL;
```

} int sys_sem_unlink(const char* name) { int i = 0; char name_buf[20]; while ((name_buf[i] = get_fs_byte(name + i)) != '\0')

```
    i++;
```

for (i = 0; i < NR_SEM; i++) {

```
    if (_str_compare(sem_list[i].name, name_buf) == 0)
    {
        (sem_list[i].name)[0] = '\0';
        sem_list[i].p_wait_queue = NULL;
        return 0;
    }
```

}

return -1; } void sem_init() { int i; for (i = 0; i < NR_SEM; i++) {

```
    (sem_list[i].name)[0] = '\0';
```

} for (i = 0; i < NR_WAIT_NODE; i++) {

```
    (wait_node_list[i]).idle = 1;
```

} return; } int sys_sem_wait(sem_t * sem) { wait_node_t* p, * q; if (sem == NULL)

```
    return -1;
```

cli(); sem->value--; //printk("In wait ,sem %s value is %d\n",sem->name,sem->value); if (sem->value < 0) {

```
    current->state = TASK_UNINTERRUPTIBLE;
    p = _find_idle_wait_node();
    p->p_task_struct = current;
    if (sem->p_wait_queue)
    {
        q = sem->p_wait_queue;
        while (q->next_node)
            q = q->next_node;
        q->next_node = p;
    }
    else
        sem->p_wait_queue = p;
    schedule();
```

} sti(); return 0; } int sys_sem_post(sem_t * sem) { wait_node_t* p; if (sem == NULL)

```
    return -1;
```

cli(); sem->value++; //printk("In post sem %s value is %d\n",sem->name,sem->value); if (sem->value <= 0) {

```
    /*p = sem->sem_queue[i];*/
    p = sem->p_wait_queue;
    sem->p_wait_queue = p->next_node;
    (p->p_task_struct)->state = 0;
    p->idle = 1;
    p->next_node = NULL;
```

} sti(); return 0; } ```

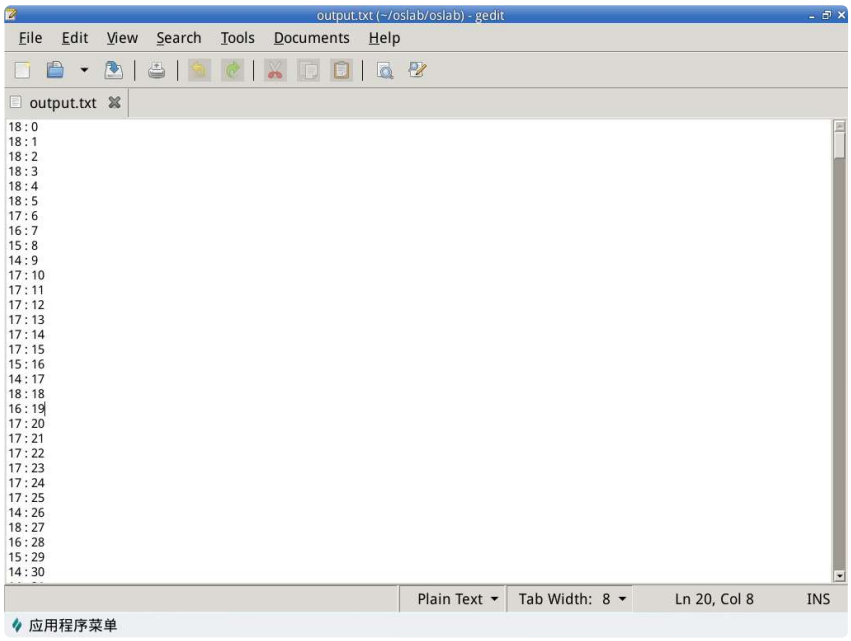
- pc.c

```c
#define __LIBRARY__
#include <linux/sem.h>
#include <unistd.h>
#include <stdio.h>
#define BUFFER_SIZE 10
#define NUMS 500
#define FORKS 5
_syscall2(sem_t*, sem_open, const char*, name, int, value);
_syscall1(int, sem_wait, sem_t*, sem);
_syscall1(int, sem_post, sem_t*, sem);
_syscall1(int, sem_unlink, const char*, name);
int main(void)
{
  FILE* fp, * fp_1;
  int num;
  int counter = 0;
  int state;
  int n, i;
  int off = 0;
  sem_t* Empty, * Full, * Mutex;

  fp_1 = freopen("output.txt", "w", stdout);
  if ((Empty = sem_open("Empty", BUFFER_SIZE)) == NULL)
      return -1;
  if ((Full = sem_open("Full", 0)) == NULL)
      return -1;
  if ((Mutex = sem_open("Mutex", 1)) == NULL)
      return -1;
  if ((fp = fopen("s_buffer", "wb+")) == NULL)
      return -1;
  /*init offset*/
  off = 0;
  fseek(fp, BUFFER_SIZE * sizeof(int), 0);
  fwrite(&off, sizeof(int), 1, fp);
  fflush(fp);
  if (!fork())
  {
      while (counter < NUMS)
      {
          sem_wait(Empty);
          sem_wait(Mutex);
          num = counter;
          fseek(fp, (counter % BUFFER_SIZE) * sizeof(int), 0);
          fwrite(&num, sizeof(int), 1, fp);
          fflush(fp);
          counter++;
          sem_post(Mutex);
          sem_post(Full);
      }
      return 0;
  }
  for (n = 0; n < FORKS; n++)
  {
      if (!fork())
      {
          for (i = 0; i < NUMS / FORKS; i++)
          {
              sem_wait(Full);
              sem_wait(Mutex);
              fseek(fp, BUFFER_SIZE * sizeof(int), 0);
              fread(&off, sizeof(int), 1, fp);
              fseek(fp, off * sizeof(int), 0);
              fread(&num, sizeof(int), 1, fp);
              printf("%d : %d\n", getpid(), num);
              /*the counter++*/
              fseek(fp, BUFFER_SIZE * sizeof(int), 0);
              off = (off + 1) % BUFFER_SIZE;
              fwrite(&off, sizeof(int), 1, fp);
              fflush(stdout);
              fflush(fp);
              sem_post(Mutex);
              sem_post(Empty);
          }
          return 0;
      }
  }
  while (wait(&state) > 0);
  fclose(fp);
  fclose(fp_1);
  if (sem_unlink("Empty") < 0)
      return -1;
  if (sem_unlink("Full") < 0)
      return -1;
```

```
  if (sem_unlink("Mutex") < 0)
      return -1;
  return 0;
}
```

5. 实验结果



6. 实验问题

不行，先判断mutex，容易引发死锁。

1

Markdown 语法

请输入想说的话

0 / 2000

发表评论

最新评论

/