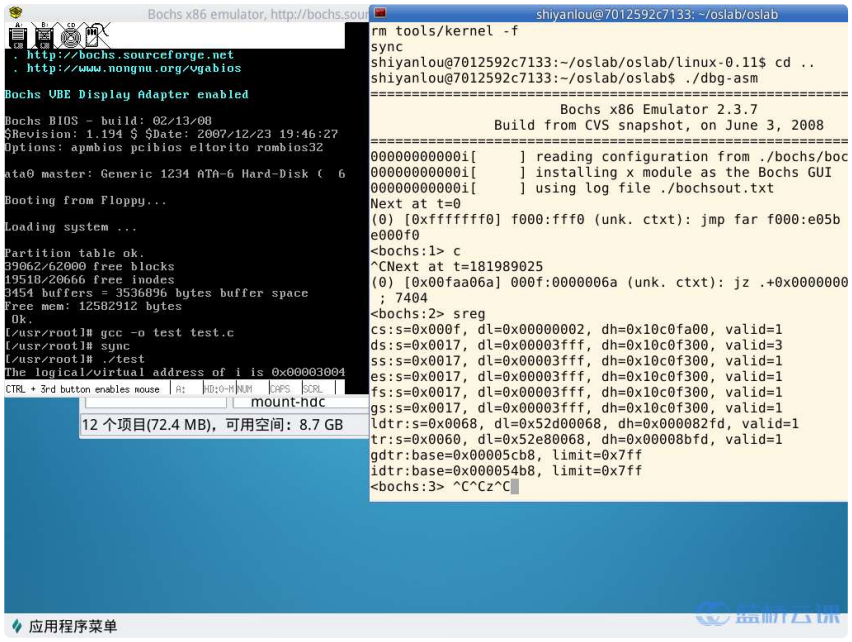
183975 L91 2018-02-18 20:40:12 地址映射与共享

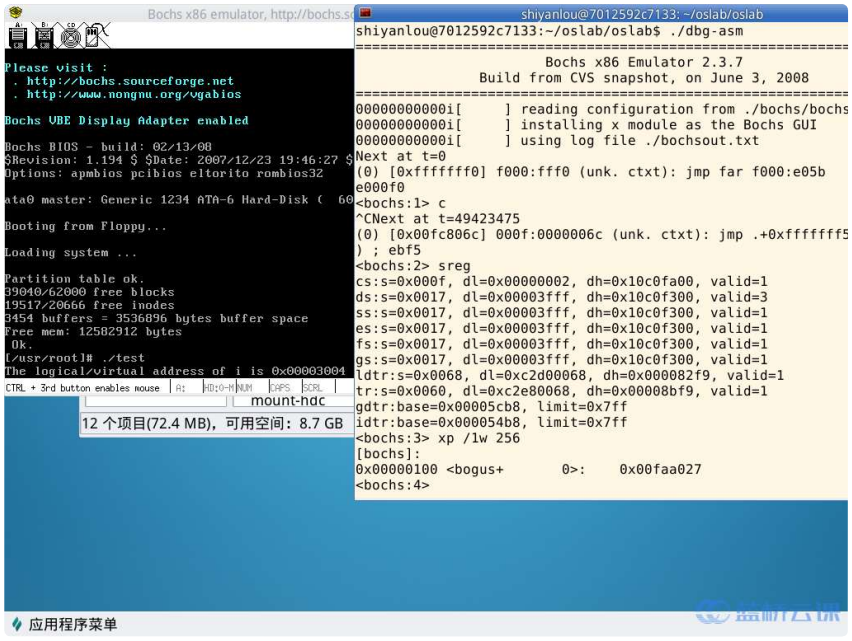
“操作系统原理与实践”实验报告

地址映射与共享

一.跟踪地址翻译过程



由图可得线性地址为：0x10000000+0x3004 = 0x10003004，可知页目录索引为256，页表索引为3，页内偏移为4



查页目录表得到页表地址：0x00faa000

实验数据

学习时间	190分钟
操作时间	117分钟
按键次数	2618次
实验次数	3次
报告字数	7221字
是否完成	完成

评分

未评分

下一篇

篇

相关报告

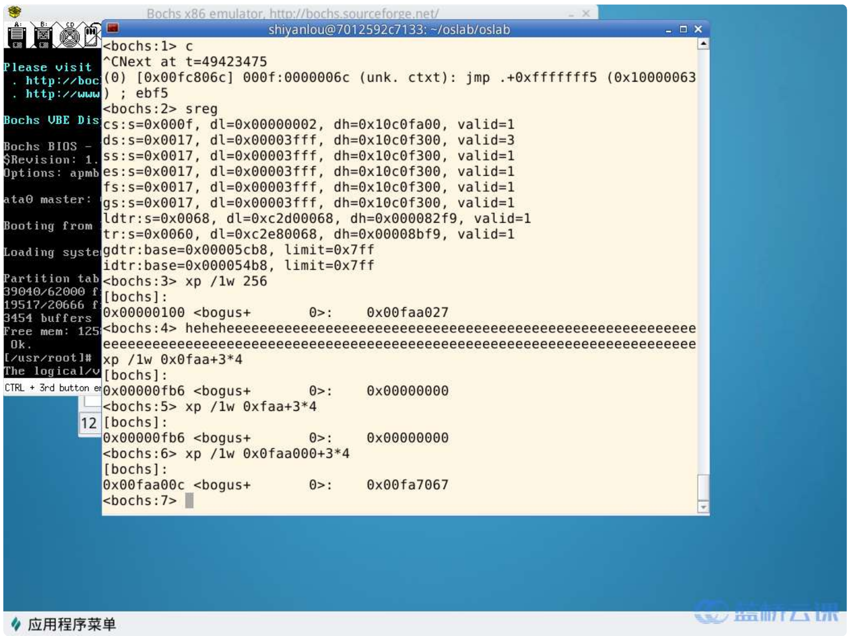
操作系统原理与实践: 熟悉实验环境 实验报告

操作系统原理与实践: 熟悉实验环境 实验报告

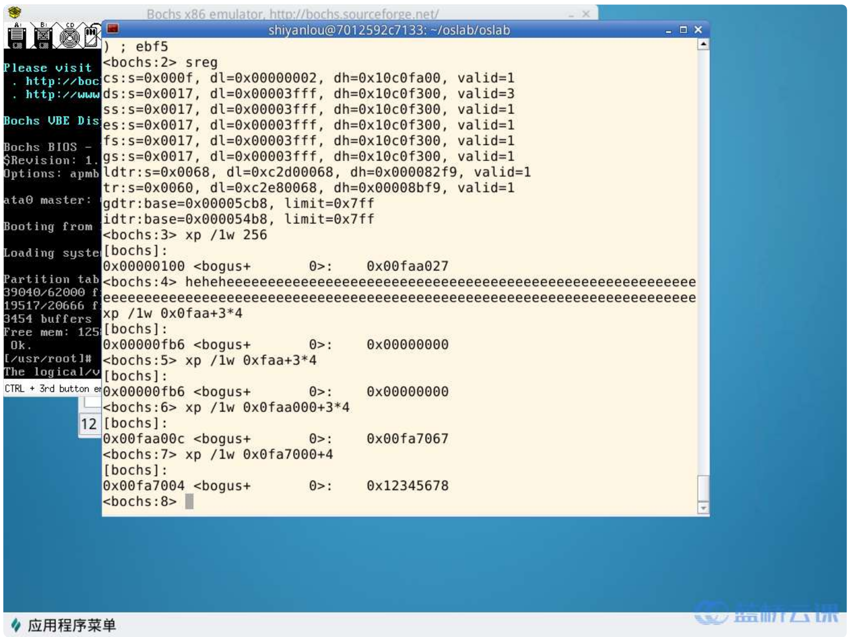
操作系统原理与实践: 基于内核栈切换的进程切换 实验报告

操作系统原理与实践: 熟悉实验环境 实验报告

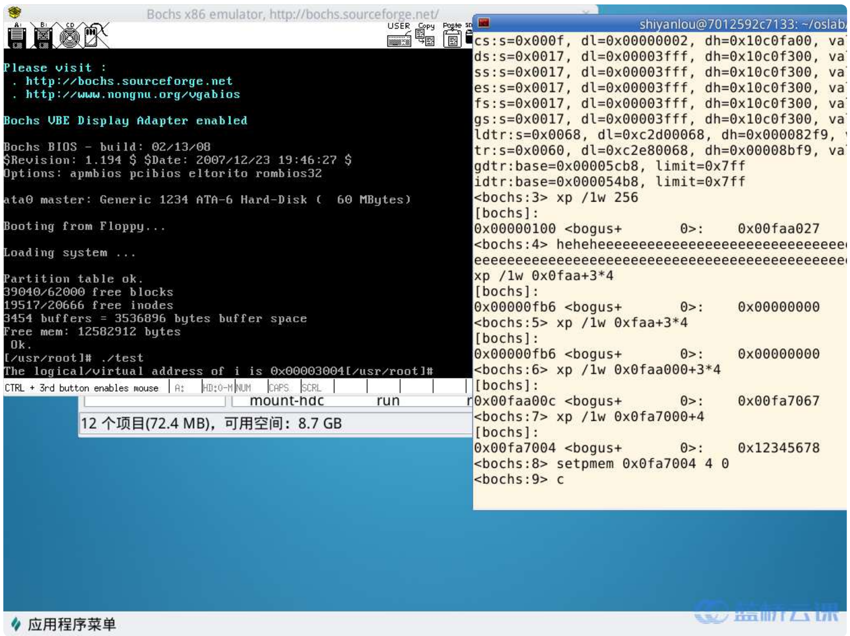
操作系统原理与实践: 信号量的实现和应用 实验报告



查页表得到页地址为：0x00fa7000



看到了变量i所在地址的内容



修改内容为0后，循环退出了

二.基于共享内存的生产者—消费者程序

1.producer.c

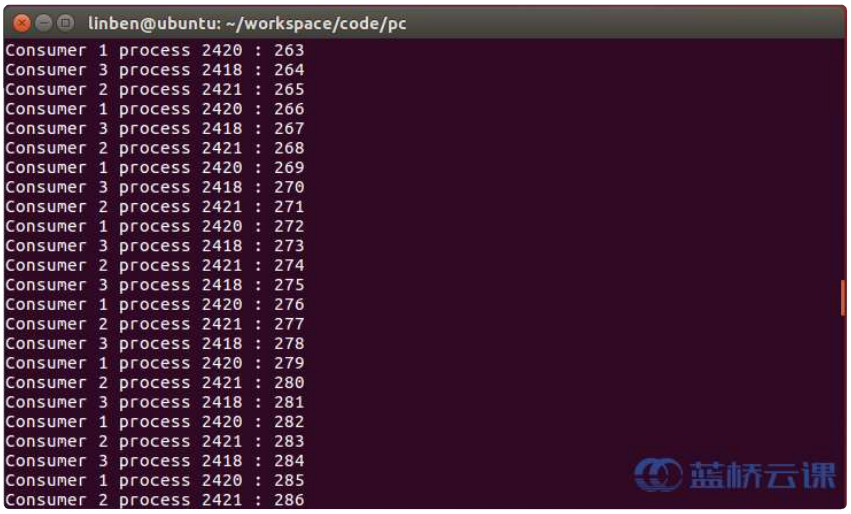
```c
#include <unistd.h>
/* constant */
#include <syscall.h>
#include <sys/shm.h>
#include <semaphore.h>
/* constant */
#include <fcntl.h>
/* printf() fflush() */
#include <stdio.h>

#define BUF_SIZE 10
#define COUNT 500
#define KEY 183
#define SHM_SIZE (BUF_SIZE+1)*sizeof(short)

int main(int argc, char** argv)
{
    int pid;
    unsigned short count = 0;
    int shm_id;
    short *shmp;
    sem_t *empty, *full, *mutex;

    shm_id = shmget(KEY, SHM_SIZE, IPC_CREAT|0666);
    if (shm_id == -1) {
        printf("shmget error\n");
        return -1;
    }
    /* managed by system, read and write */
    shmp = (short*)shmat(shm_id, NULL, 0);

    /* close old sem */
    sem_unlink("empty");
    sem_unlink("full");
    sem_unlink("mutex");
    /* open new sem */
    empty = sem_open("empty", O_CREAT|O_EXCL, 0666, 10);
    full = sem_open("full", O_CREAT|O_EXCL, 0666, 0);
    mutex = sem_open("mutex", O_CREAT|O_EXCL, 0666, 1);
    if (empty == SEM_FAILED || full == SEM_FAILED || mutex == SEM_FAILED)
printf("sem_open error\n");

    pid = syscall(SYS_getpid);
    while (count <= COUNT) {
        sem_wait(empty);
        sem_wait(mutex);

        printf("Producer 1 process %d : %d\n", pid, count);
        fflush(stdout);
        *(shmp++) = count++;
        if (!(count % BUF_SIZE)) shmp -= 10;

        sem_post(mutex);
        sem_post(full);
    }
    return 0;
}
```

2.consumer.c

```c
#include <unistd.h>
/* constant */
#include <syscall.h>
#include <sys/shm.h>
#include <semaphore.h>
/* printf() fflush() */
#include <stdio.h>

#define BUF_SIZE 10
#define KEY 183

int main(int argc, char** argv)
{
    int pid;
    int shm_id;
    short *shmp, *index;
    sem_t *empty, *full, *mutex;

    shm_id = shmget(KEY, 0, 0);
    if (shm_id == -1) {
        printf("shmget error\n");
        return -1;
    }
    /* managed by system, read and write */
    shmp = (short*)shmat(shm_id, NULL, 0);
    index = shmp+BUF_SIZE;
    *index = 0;

    /* open old sem */
    empty = sem_open("empty", 0);
    full = sem_open("full", 0);
    mutex = sem_open("mutex", 0);
    if (empty == SEM_FAILED || full == SEM_FAILED || mutex == SEM_FAILED)
printf("sem_open error\n");

    if (!syscall(SYS_fork)) {
        pid = syscall(SYS_getpid);
        while (1) {
            sem_wait(full);
            sem_wait(mutex);

            printf("Consumer 1 process %d : %d\n", pid, shmp[*index]);
            fflush(stdout);
            if (*index == 9)
                *index = 0;
            else
                (*index)++;

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

    if (!syscall(SYS_fork)) {
        pid = syscall(SYS_getpid);
        while (1) {
            sem_wait(full);
            sem_wait(mutex);

            printf("Consumer 2 process %d : %d\n", pid, shmp[*index]);
            fflush(stdout);
            if (*index == 9)
                *index = 0;
            else
                (*index)++;

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

    pid = syscall(SYS_getpid);
    while (1) {
        sem_wait(full);
        sem_wait(mutex);

        printf("Consumer 3 process %d : %d\n", pid, shmp[*index]);
        fflush(stdout);
        if (*index == 9)
            *index = 0;
        else
```

```
            (*index)++;

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

3.结果



三.共享内存的实现

主要是2个系统调用函数，生产者消费者程序在Ubuntu上好像没触发写时复制，在linux上肯定会的，所以需做些调整。

1.shm.c

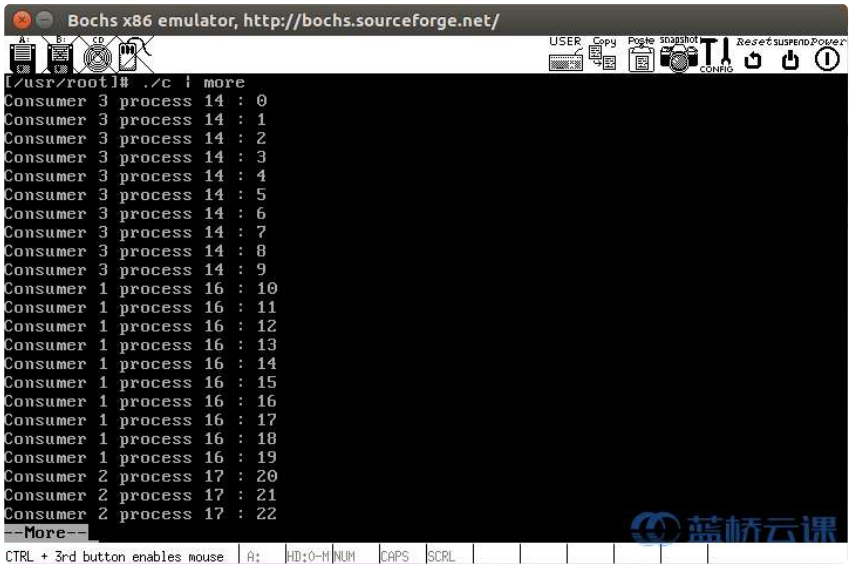
```
#include <sys/shm.h>
/* printk() */
#include <linux/kernel.h>
#include <errno.h>
/* get_free_page() PAGE_SIZE */
#include <linux/mm.h>
#include <linux/sched.h>

shm_t shms[SHMS_SIZE] = {{0, 0, 0}};

int sys_shmget(key_t key, size_t size)
{
    int i;

    /* get old shm */
    for (i=0; i<SHMS_SIZE; i++)
        if (shms[i].key == key) return i;

    /* creat new shm */
    for (i=0; i<SHMS_SIZE && shms[i].key; i++) ;
    if (i == SHMS_SIZE) {
        printk("no shm place");
        return -ENOMEM;
    }
    if (size > PAGE_SIZE) return -EINVAL;
    shms[i].size = size;
    shms[i].key = key;
    if (!(shms[i].addr = get_free_page())) return -ENOMEM;
    return i;
}

void* sys_shmat(int shmid)
{
    unsigned long logical_addr;

    if (shmid < 0 || shmid >= SHMS_SIZE) return -EINVAL;
    logical_addr = current->brk;
    current->brk +=  PAGE_SIZE;
    if (!put_page(shms[shmid].addr, current->start_code+logical_addr)) return -
ENOMEM;
    return (void*)logical_addr;
}
```

2.consumer.c

```c
#define __LIBRARY__
#include <unistd.h>
#include <sys/shm.h>
#include <sys/sem.h>
/* printf() fflush() */
#include <stdio.h>

_syscall0(int, fork);
_syscall0(int, getpid)
_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)
_syscall2(int, shmget, key_t, key, size_t, size)
_syscall1(void*, shmat, int, shmid)

#define BUF_SIZE 10
#define KEY 183

int main(int argc, char** argv)
{
    int pid;
    int shm_id;
    short *shmp, *index;
    sem_t *empty, *full, *mutex;

    shm_id = shmget(KEY, 0);
    if (shm_id == -1) {
        printf("shmget error\n");
        return -1;
    }

    /* open old sem */
    empty = sem_open("empty", 0);
    full = sem_open("full", 0);
    mutex = sem_open("mutex", 0);
    if (empty == -1 || full == -1 || mutex == -1) printf("sem_open error\n");

    /* consumer 1 */
    if (!fork()) {
        /* managed by system, read and write */
        shmp = (short*)shmat(shm_id);
        if (shmp == -1)
            printf("sem_open error\n");
        index = shmp+BUF_SIZE;
        *index = 0;
        pid = getpid();
        while (1) {
            sem_wait(full);
            sem_wait(mutex);

            printf("Consumer 1 process %d : %d\n", pid, shmp[*index]);
            fflush(stdout);
            if (*index == BUF_SIZE-1)
                *index = 0;
            else
                (*index)++;

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

    /* consumer 2 */
    if (!fork()) {
        /* managed by system, read and write */
        shmp = (short*)shmat(shm_id);
        if (shmp == -1)
            printf("sem_open error\n");
        index = shmp+BUF_SIZE;
        *index = 0;
        pid = getpid();
        while (1) {
            sem_wait(full);
            sem_wait(mutex);

            printf("Consumer 2 process %d : %d\n", pid, shmp[*index]);
            fflush(stdout);
            if (*index == BUF_SIZE-1)
                *index = 0;
            else
                (*index)++;
```

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

    /* consumer 3 */
    /* managed by system, read and write */
    shmp = (short*)shmat(shm_id);
    if (shmp == -1)
        printf("sem_open error\n");
    index = shmp+BUF_SIZE;
    *index = 0;
    pid = getpid();
    while (1) {
        sem_wait(full);
        sem_wait(mutex);

        printf("Consumer 3 process %d : %d\n", pid, shmp[*index]);
        fflush(stdout);
        if (*index == BUF_SIZE-1)
            *index = 0;
        else
            (*index)++;

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

3.结果



四.回答问题

完成实验后，在实验报告中回答如下问题：

- 对于地址映射实验部分，列出你认为最重要的那几步（不超过4步），并给出你获得的实验数据。
- test.c退出后，如果马上再运行一次，并再进行地址跟踪，你发现有哪些异同？为什么？

1.通过GDT找到LDT，通过LDT找到进程基址，通过页目录中找到页表，通过页表找到页 2.没做，估计物理内存可能会变，线性地址不变。因为使用同一进程数组项，但因为前一次运行占用了一页内存。

0 / 2000

发表评论

最新评论