OSLAB4实验报告

余晨宁 151242062 匡亚明学院
151242062@smail.nju.edu.cn
2017.05.30

问题回答

一、什么地方是所有进程共享的呢？你可以据此实现你的进程匿名信号量么?

答：物理内存是所有进程共享的地方。

申请信号量的时候在物理内存中申请一块存储该信号量的空间，并映射到申请该信号量的进程的虚拟空间中，之后映射到这个进程所有的子进程当中，通过读取并修改该信号量来实现PV操作，通过将记录这个信号量所处空间是否被占用的标记置为false来destroy。

二、普通的全局变量是不是可以被用户修改?

答：如果这个全局变量是多个线程/进程共享的，那么我认为是否可以被修改需要看这个全局变量是否允许多个线程/进程同时修改。如果允许，那么可以修改。如果不允许，需要通过信号量操作来得到修改权限。

三、linux进程和线程下匿名信号量的实现有什么本质区别?

答：线程信号量被存储在用户空间中，进程信号量被存储在内核空间中。

四、结合你的os，你该怎样实现进程/线程的具名信号量？（可以纸上谈兵）

答：想法和第一题类似。申请信号量的时候在物理内存中申请一块存储该信号量的空间，并映射到申请该信号量的进程的虚拟空间中，且在所有申请同样名字的进程/线程中创建映射，使对应的虚拟空间映射到相同的物理内存，通过读取并修改该信号量来实现PV操作，通过将记录这个信号量所处空间是否被占用的标记置为false来destroy。

实验环境

ubuntu：64位16.04，gcc：5.3.1, 要先apt-get install gcc-4.8-multilib

实验进度

修复了上次fork_test无限输出“X”而不是只输出6个”X”的巨大bug。
实现了sem_init, sem_wait, sem_post, sem_destroy, sem_trywait, create_thread等函数。

模仿书上完成了producer_consumer问题的代码，并把其放在了game里。

实验心得和想法

在认清自己的分配内存的方式很乱之后，借鉴了github上ShijianXu学长的代码结构，相当于重写了一遍oslab3，这告诉我们代码的整理很重要。

producer_consumer的代码如右图，

sem_init的第一个参数告诉了sem_t变量的地址，第二个参数告诉了想要设置的初始值，第三个参数告诉了是否是mutex。如果是mutex，则为1。

create_thread的参数为函数的指针。

这次实验比较简单（比上次的fork实现简单到不知道哪里去了）。我是通过在pcb中记录每个进程记录使其block的semaphore，当V操作进行时，若能释放权限，则在对应的所有被对应的semaphore block住的进程中释放一次。和notifyAll想法一致。

在实现原子操作的时候，我差点忘了先关中断，这一点有些惊险。

没啥惊天bug。

通过这次实验，我进一步理解了信号量机制和线程的创建，有所收获。

```
#define item_num 10 //缓冲区的数目

int in ;        //where the producer put the item
int out;        //where the consumer take the item
int items[item_num];
sem_t empty;        //if the items is full, 'empty' blocks producer
sem_t full;     //if the items is empty, 'full' blocks producer
sem_t mutex;

void print()
{
    int i;
    for(i = 0; i < item_num; i ++)
        printf("%d ", items[i]);
    printf("\n");
}

void producer()
{
    while(1)
    {
        sem_wait(&empty);
        sem_wait(&mutex);
        in = in % item_num;
        printf("producer:\n", in);
        items[in] = 1;
        print();
        in++;
        sem_post(&mutex);
        sem_post(&full);
    }
}

void consumer()
{
    while(1)
    {
        sem_wait(&full);
        sem_wait(&mutex);
        out = out % item_num;
        printf("consumer:\n", out);
        items[out] -= 1;
        print();
        out++;
        sem_post(&mutex);
        sem_post(&empty);
    }
}

void pc_main()
{
    in = 0;
    out = 0;
    memset(items, 0, sizeof(items));
    sem_init(&mutex, 1, 1);
    sem_init(&empty, item_num, 0);
    sem_init(&full, 0, 0);
    printf("sem_init success\n");
    create_thread((uint32_t *)producer);
    create_thread((uint32_t *)consumer);
    while(1);
}
```