

操作系统 Lab4——进程同步

151220045 蒋雨霖

一. 实验目的:

通过实现一个简单的生产者消费者程序,介绍基于信号量的进程同步机制

二. 实验思路

1. 定义信号量的结构体, 在之前实现的 PCB 表中添加一个信号量的指针;

```
struct Semaphore
{
    int value;
    struct ProcessTable *block;
};
```

2. sem_init 系统调用用于初始化信号量, 其中参数 value 用于指定信号量的初始值, 初始化成功则返回 0, 指针 sem 指向初始化成功的信号量。

```
void sem_init(uint32_t *sem, uint32_t value)
{
    // sem_free=0;
    st[*sem].value=value;
    rn->sem=(struct Semaphore*)&(st[(*sem)++]);
}
```

3. 实现 PV 操作:

sem_post 系统调用对应信号量的 V 操作, 其使得 sem 指向的信号量的 value 增一, 若 value 取值不大于 0, 则释放一个阻塞在该信号量上进程 (即将该进程设置为就绪态), 若操作成功则返回 0, 否则返回-1.:

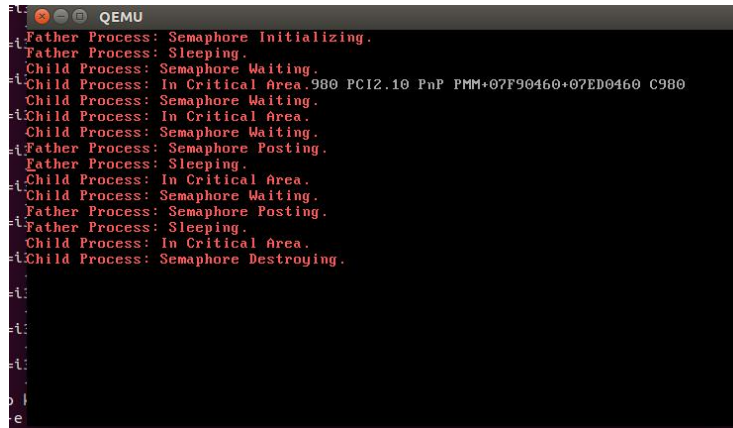
```
void V(Semaphore* sem)
{
    sem->value++;
    if(sem->value<=0)
    {
        ProcessTable* re=dequeue(&(sem->block));
        insert(re);
    }
}
```

sem_wait 系统调用对应信号量的 P 操作, 其使得 sem 指向的信号量的 value 减一, 若 value 取值小于 0, 则阻塞自身, 否则进程继续执行, 若操作成功则返回 0, 否则返回-1

```
void P(Semaphore* sem)
{
    sem->value--;
    if(sem->value<0)
    {
        enqueue(&(sem->block), rn);
        rn=NULL;
        schedule();
    }
}
```

4. 在 `syscall.c` 中分别实现几个要求的系统调用函数, 在 `irqHandle` 中增加了 5、6、7、8 调用号, 并且 `fork` 中需要做小小修改, 将信号量指针的传递赋值也加入。 `sem_t` 我定义为 `uint32_t` 通过 `ecx` 传递到内核, 决定当前为哪个信号量。

三. 实验效果截图:



```
QEMU
Father Process: Semaphore Initializing.
Father Process: Sleeping.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.980 PCI2.10 PnP PMM+07F90460+07ED0460 C980
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Child Process: In Critical Area.
Child Process: Semaphore Destroying.
```

四. 实验心得:

这次实验相比前几次实验简单了许多, 完成起来轻松了许多。