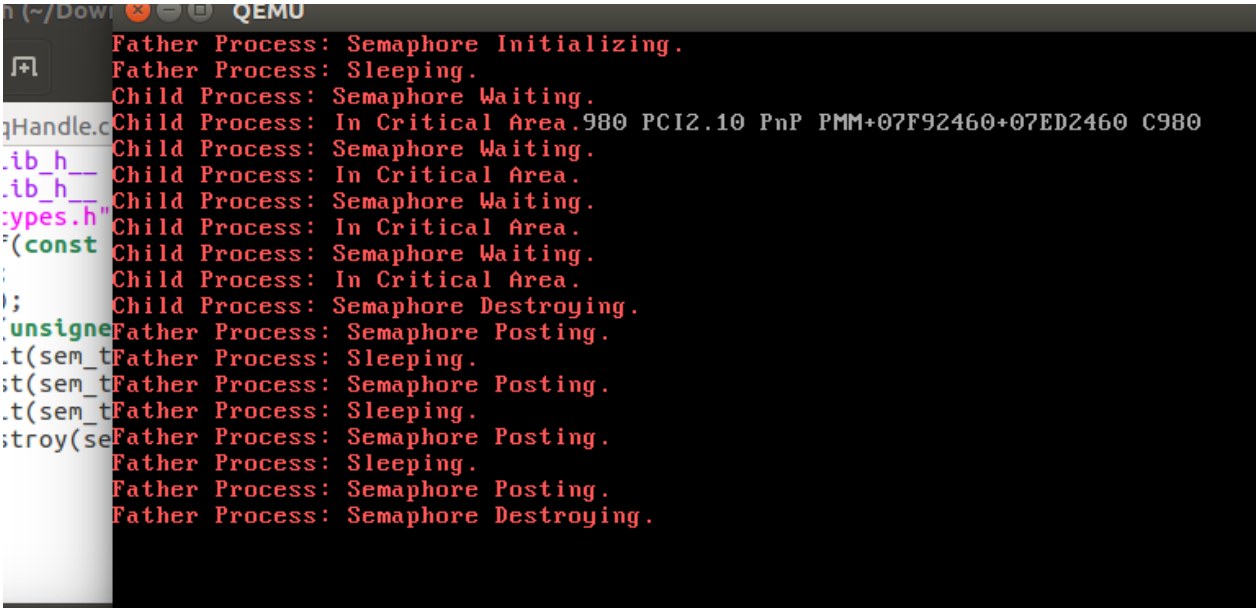
Lab4 进程同步——实验报告

151220098 汤聪

一. 实验要求：

本实验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. 实验结果：



三. 实验过程：

（1） 信号量机制

先定义一个.h 文件，定义结构体 semaphore

```
typedef struct Semaphore
{
        int value;
        int id;
        int ifuse;
        PCB *next;
}Semaphore;
```

Value 表示信号量，id 表示序号，next 表示进程链表

定义 PV 操作函数，定义 WR 函数

先将信号量结构体初始化，然后调用 PV 操作

w 函数用于阻塞进程自身在该信号量上，如果 s->next 为空，将 current 指向

pcb_runnable 队列，如果不为空，则将 current 置为阻塞态，将 s 加入 pcb_runnable 队列

```
if(s->next==NULL)
{
        s->next=current;
        current->preIndex=-1;
        current->nextIndex=-1;
        current->state=BLOCKED;
        if(pcb_runnable==NULL)
                return;
        else
        {
                current=pcb_runnable;
                if(pcb_runnable->nextIndex==-1)
                        pcb_runnable=NULL;
                else
                {
                pcb_runnable=&pcb[pcb_runnable->nextIndex]
                pcb_runnable->preIndex=-1;
                }
        }
        if(current==NULL)
                current=&idle;
}
else
{
        PCB *temp=s->next;
        int index=temp->pid;
        while(pcb[index].nextIndex!=-1)
                index=pcb[index].nextIndex;
        current->preIndex=index;
        pcb[index].nextIndex=current->pid;
        current->state=BLOCKED;
        current->preIndex=-1;
        current->nextIndex=-1;
        if(pcb_runnable==NULL)
                return;
        else
        {
                current=pcb_runnable;
                if(pcb_runnable->nextIndex==-1)
                        pcb_runnable=NULL;
                else
```

R 函数用于释放一个阻塞在该信号量上的进程，将信号量加入 pcb_runnable 队列，并将其从元队列中删除

```
    if(s->next==NULL)
            return;
    else
    {
            PCB *temp=s->next;
            int m=temp->pid;
            int k=pcb[m].nextIndex;
            pcb[k].preIndex=-1;
            s->next->pid=k;
            temp->state=RUNNABLE;
            temp->preIndex=-1;
            temp->nextIndex=-1;
            int index=pcb_runnable->pid;
            while(pcb[index].nextIndex!=-1)
                    index=pcb[index].nextIndex;
            temp->preIndex=index;
            pcb[index].nextIndex=current->pid;
    }
```

(2) 系统调用

Sem_init 系统调用用于初始化信号量，参数 value 用于指定信号量的初始值，sem 指针指向初始化成功的信号量

```
int sem_init(sem_t *sem,uint32_t value)
{
        sem->value = value;
        sem->id = syscall(6,0,value,0,0,0);
        if(sem->id==-1)
                return -1;
        else
                return 0;
}
```

Sem_post 系统调用对应于 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

```
int sem_wait(sem_t *sem)
{
        int ret=syscall(8,0,sem->id,0,0,0);
        if(ret==-1)
                return -1;
        else
        {
                sem->value=ret;
                return 0;
        }
}
```

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

```
int sem_wait(sem_t *sem)
{
        int ret=syscall(8,0,sem->id,0,0,0);
        if(ret==-1)
                return -1;
        else
        {
                sem->value=ret;
                return 0;
        }
}
```

sem_destroy 系统调用用于销毁 sem 指向的信号量，销毁成功则返回 0，否则返回-1，若尚有进程阻塞在该信号量上，可带来未知错误

```
int sem_destroy(sem_t *sem)
{
        int ret=syscall(9,0,sem->id,0,0,0);
        if(ret==-1)
                return -1;
        else
                return 0;
}
```

在 irqHandle 中增加上述系统调用，匹配相应的调用路径即可。