

# 操作系统实验四报告

161220149 徐安孜

## 一、数据结构

```
struct Semaphore{
    int used;
    int value;
    int list[5];
};
typedef struct Semaphore Semaphore;
struct Semaphore Sem[10];
```

在内核维护 Semaphore 这一数据结构，其中成员 value 为此信号量的值；used 记录此信号量是否被使用，若未被使用则为 0，否则为 1；list 中存放阻塞在此信号量上的进程的 pid，初始时 list 中值为-1。

定义 Semaphore 类型长度为 10 的数组，也就是说共有 10 个信号量可供使用。

## 二、几个系统调用实现

### (1) sem\_init()

```
int sem_init(struct TrapFrame *tf){
    int i=0;
    for(;Sem[i].used!=0;i++);
    if(i>=10)
        return -1;
    Sem[i].used=1;
    Sem[i].value=tf->ecx;
    for(int j=0;j<5;j++)
        Sem[i].list[j]=-1;
    return i;
}
```

在 Sem 数组中找 used 为 0 的数组元素，若为找到表明没有可供使用的信号量，返回-1；若找到未用信号量，则初始化其 value、list 返回其下标 i

### (2) sem\_post()

```
int sem_post(struct TrapFrame *tf){
    if(tf->ecx>=10 || tf->ecx<0)return -1;
    if(Sem[tf->ecx].used==0)return -1;

    struct Semaphore *sem=&Sem[tf->ecx];
    sem->value++;
    int i=0;
    if(sem->value<=0){
        for(;sem->list[i]==-1 && i<5;i++){
            if(i<5){
                PCB[sem->list[i]].state=STATE_RUNNABLE;
                PCB[sem->list[i]].timeCount=1;
                sem->list[i]=-1;
            }
        }
    }
    return 0;
}
```

tf->ecx 存放的是所要操作的信号量在 Sem 数组中的下标

若此下标不在 0-9 范围内，或此下标对信号量未被使用，则发生错误，返回-1

否则，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）

### (3)sem\_wait()

```
int sem_wait(struct TrapFrame *tf){
    if(tf->ecx>=10 || tf->ecx<0)return -1;
    if(Sem[tf->ecx].used==0)return -1;

    struct Semaphore *sem=&Sem[tf->ecx];
    sem->value--;
    if(sem->value<0){
        sem->list[current]=current;
        PCB[current].timeCount=-1;
        PCB[current].state=STATE_BLOCKED;
        schedule();
    }
    return 0;
}
```

与 sem\_post()类似

### (4)sem\_destroy()

```
int sem_destroy(struct TrapFrame *tf){
    if(tf->ecx>=10 || tf->ecx<0)return -1;
    if(Sem[tf->ecx].used==0)return -1;

    Sem[tf->ecx].used=0;
    return 0;
}
```

将 tf->ecx 所指的信号量 used 置为 0;