OSLab3

191220083 蒙芷露 mzl0830@sina.com

整体思路：利用系统调用来调用实现好的关于进程创建、切换、撤销的函数；重点在于pcb的维护和栈信息的更新与维护

3.1 syscall.c

syscall.c中三个函数需要依赖syscall调用的底层函数，所以放在最后填写

3.2 时钟中断处理：

timerHandle中进程切换的实现 以及 对于手册提供部分代码段的理解

```
pcb[current].timeCount++;
    if(pcb[current].timeCount>MAX_TIME_COUNT)
    {
        pcb[current].state=STATE_RUNNABLE;
        pcb[current].timeCount=0;
        int i=0;
        //Round Robin
        for(i=(current+1)%MAX_PCB_NUM;i!=current;i++)
        {
            if(pcb[i].state==STATE_RUNNABLE)
                break;
        }
        current=i;
        pcb[current].state=STATE_RUNNING;
    }
tmpStackTop = pcb[current].stackTop;
    //set tss for user process
    pcb[current].stackTop = pcb[current].prevStackTop;//把之前保持的栈信息装载
    tss.esp0 = (uint32_t)&(pcb[current].stackTop);//把当前栈顶信息装载进入tss的esp0
    //switch to kernel
    asm volatile("movl %0, %%esp"::"m"(tmpStackTop)); // switch kernel stack
    asm volatile("popl %gs");
    asm volatile("popl %fs");
    asm volatile("popl %es");
    asm volatile("popl %ds");
    asm volatile("popal");
    asm volatile("addl $8, %esp");
    asm volatile("iret");//依次压入栈的信息并返回
```

3.3 系统调用例程

先补充syscallHandle中的调用：

```
void syscallHandle(struct StackFrame *sf) {
    switch(sf->eax) { // syscall number
        case 0:
            syscallWrite(sf);
            break; // for SYS_WRITE
```

```
        /*TODO Add Fork,Sleep... */
        case 1:
            syscallFork(sf);
            break; // SYS_FORK
        case 3:
            syscallSleep(sf);
            break;//SYS_SLEEP
        case 4:
            syscallExit(sf);
            break;//SYS_EXIT
        default:break;
    }
}
```

syscallFork

```
//分配pcb块 找到空闲的 空闲是之前的已经dead
for(i=0;i<MAX_PCB_NUM;++i)
    {
        //putChar('0'+pcb[i].state);
        //putChar('\n');
        if(pcb[i].state==STATE_DEAD)
            break;
    }
//代码段和数据段的完全拷贝 我们默认每个pcb对应进程的内存空间固定， pcb[i] 对应的内存起始地址为
(i + 1) * 0x100000 ， 大小为 0x100000
if(i!=MAX_PCB_NUM)//has resource for fork
    {
        enableInterrupt();
        for(j=0;j<0x100000;++j)
        {
            *(uint8_t*)(j+(i+1)*0x100000)=*(uint8_t*)(j+(current+1)* 0x100000);
            if(j%0x1000==0)
                asm volatile("int $0x20"); //XXX Testing irqTimer during syscall
        }//copy the data and code
        disableInterrupt();
    //pcb属性：直接复制 计算 与父进程无关 参考了initProc
    for(j=0;j<sizeof(ProcessTable);++j)
            *((uint8_t*)(&pcb[i])+j)=*((uint8_t *)(&pcb[current])+j);
        //copy pcb info
        pcb[i].stackTop = (uint32_t)&(pcb[i].regs);
        pcb[i].prevStackTop = (uint32_t)&(pcb[i].stackTop);
        pcb[i].state = STATE_RUNNABLE;
        pcb[i].timeCount = 0;
        pcb[i].sleepTime = 0;
        pcb[i].pid = i;
        //set regs
        pcb[i].regs.ss = USEL(2+2*i);
        pcb[i].regs.cs = USEL(1+2*i);
        pcb[i].regs.ds = USEL(2+2*i);
        pcb[i].regs.es = USEL(2+2*i);
        pcb[i].regs.fs = USEL(2+2*i);
        pcb[i].regs.gs = USEL(2+2*i);
        //set return value
        pcb[i].regs.eax=0;
        pcb[current].regs.eax=i;
        putChar('F');putChar('o');putChar('r');putChar('k');
```

```
        putChar('0' + pcb[i].pid);putChar('\n');
    }
//返回值 用eax返回
else
        pcb[current].regs.eax=-1;

    return ;//return to user space
```

syscallSleep:

```
int i=0;
pcb[current].state=STATE_BLOCKED;//当前进程状态BLOCKED
pcb[current].sleepTime=sf->ecx;//sleepTime设置为传入的参数
//模拟时钟中断做切换
for(i=(current+1)%MAX_PCB_NUM;i!=current;i=(i+1)%MAX_PCB_NUM)
        if(pcb[i].state==STATE_RUNNABLE)
            break;

    current = i;
    pcb[current].state = STATE_RUNNABLE;

    //recover stackTop of selected process
    uint32_t tmpStackTop = pcb[current].stackTop;
    pcb[current].stackTop=pcb[current].prevStackTop;
    //set tss for user process
    tss.esp0=pcb[current].stackTop;
    tss.ss0=KSEL(SEG_KDATA);

    //switch to kernel stack
    asm volatile("movl %0, %%esp" : :"m"(tmpStackTop));
    asm volatile("popl %gs");
    asm volatile("popl %fs");
    asm volatile("popl %es");
    asm volatile("popl %ds");
    asm volatile("popal");
    asm volatile("addl $8, %esp");
    asm volatile("iret");
    return;
```

syscallExit

```
//当前进程状态变成DEAD
    int i=0;
    pcb[current].state=STATE_DEAD;
//模拟时钟中断做切换
    for(i=(current+1)%MAX_PCB_NUM;i!=current;i=(i+1)%MAX_PCB_NUM)
            if(pcb[i].state==STATE_RUNNABLE)
                break;
    current = i;
    pcb[current].state=STATE_RUNNING;
    uint32_t tmpStackTop = pcb[current].stackTop;
    pcb[current].stackTop=pcb[current].prevStackTop;
    //set tss for user process
    tss.esp0=pcb[current].stackTop;
    tss.ss0=KSEL(SEG_KDATA);
```

```
    //switch to kernel stack
    asm volatile("movl %0, %%esp" : :"m"(tmpStackTop));
    asm volatile("popl %gs");
    asm volatile("popl %fs");
    asm volatile("popl %es");
    asm volatile("popl %ds");
    asm volatile("popal");
    asm volatile("addl $8, %esp");
    asm volatile("iret");
    return;
```

完成之后到syscall.c中填写库函数的实现

3.4 中断嵌套

关键是在函数运行的过程中允许中断打开

最开始按照手册的添加，屏幕出现无法显示的情况，应该是因为一次进程创建中有太多次中断嵌套运行变得太慢，于是对代码进行了修改：

```
        enableInterrupt();
        for(j=0;j<0x100000;++j)
        {
            *(uint8_t*)(j+(i+1)*0x100000)=*(uint8_t*)(j+(current+1)* 0x100000);
            if(j%0x1000==0)
                asm volatile("int $0x20"); //XXX Testing irqTimer during syscall
        }//copy the data and code
        disableInterrupt();
```

开启中断嵌套后仍然可以正常运行：