lab3 进程切换

191220138 杨飞洋

实验进度

已完成所有基本内容，选做部分完成了中断嵌套。

特别注意

并且我对框架代码做出了一定的修改，不然在我的机器上跑不起来。

在boot.c中的`bootMain()`函数中，将`offset`就设为`0x1000`，并且不用`phoff`，如下：

```
    //int phoff = 0x34;
    int offset = 0x1000;
    unsigned int elf = 0x100000;
    void (*kMainEntry)(void);
    kMainEntry = (void(*)(void))0x100000;

    for (i = 0; i < 200; i++) {
        readSect((void*)(elf + i*512), 1+i);
    }

    kMainEntry = (void(*)(void))((struct ELFHeader *)elf)->entry;
    //phoff = ((struct ELFHeader *)elf)->phoff;
    //offset = ((struct ProgramHeader *)(elf + phoff))->off;
```

数据结构和一些宏

syscall相关

```
#define SYS_WRITE 0
#define SYS_FORK 1
#define SYS_EXEC 2
#define SYS_SLEEP 3
#define SYS_EXIT 4

#define STD_OUT 0

#define MAX_BUFFER_SIZE 256
```

进程结构

```
#define MAX_STACK_SIZE 1024
#define MAX_PCB_NUM ((NR_SEGMENTS-2)/2)

#define STATE_RUNNABLE 0
#define STATE_RUNNING 1
#define STATE_BLOCKED 2
#define STATE_DEAD 3

#define MAX_TIME_COUNT 16

struct StackFrame {
    uint32_t gs, fs, es, ds;
    uint32_t edi, esi, ebp, xxx, ebx, edx, ecx, eax;
    uint32_t irq, error;
    uint32_t eip, cs, eflags, esp, ss;
};

struct ProcessTable {
    uint32_t stack[MAX_STACK_SIZE];
    struct StackFrame regs;
    uint32_t stackTop;
    uint32_t prevStackTop;
    int state;
    int timeCount;
    int sleepTime;
    uint32_t pid;
    char name[32];
};
typedef struct ProcessTable ProcessTable;
```

有用的全局变量

```
SegDesc gdt[NR_SEGMENTS];       // the new GDT, NR_SEGMENTS=10, defined in
x86/memory.h
TSS tss;

ProcessTable pcb[MAX_PCB_NUM]; // pcb
int current; // current process
```

实验目的

主要完成自制简单操作系统的进程管理功能，通过实现一个简单的任务调度，介绍基于时间中断进行进程切换完成任务调度的全过程，主要涉及到fork、exit、sleep等库函数和对应的处理例程实现。

实验过程

完成库函数

在syscall.c中，在相应的函数下调用`syscall()`函数即可，只要设置一下传入的参数就行了，代码如下：

```
pid_t fork() {
    return syscall(SYS_FORK, 0, 0, 0, 0, 0);
}


int sleep(uint32_t time) {
    return syscall(SYS_SLEEP, time, 0, 0, 0, 0);
}

int exit() {
    return syscall(SYS_EXIT, 0, 0, 0, 0, 0);
}
```

时钟中断处理

时钟中断功能：

1.遍历`pcb`，将状态为`STATE_BLOCKED`的进程的`sleepTime`减一，如果进程的`sleepTime`变为0，重新设为`STATE_RUNNABLE`

2.将当前进程的`timeCount`加一，如果时间片用完`(timeCount==MAX_TIME_COUNT)`且有其它状态为`STATE_RUNNABLE`的进程，切换，否则继续执行当前进程

可以根据`processtable`和`pcb,current`来完成上述逻辑，关于进程切换的代码手册中已经给出，具体实现代码如下：

```
void timerHandle(struct StackFrame *sf) {
    uint32_t tmpStackTop;
    for (int i = 0; i < MAX_PCB_NUM; i++){
        if (pcb[i].state == STATE_BLOCKED){
            pcb[i].sleepTime--;
            if (pcb[i].sleepTime == 0)
                pcb[i].state = STATE_RUNNABLE;
        }
    }
    pcb[current].timeCount++;
    if (pcb[current].timeCount >= MAX_TIME_COUNT){
        int i = (current + 1) % MAX_PCB_NUM;
        while (i != current){
            if (pcb[i].state == STATE_RUNNABLE)
                break;
            i = (i + 1) % MAX_PCB_NUM;
        }
        if (i != current){
            current = i;
            pcb[current].state = STATE_RUNNING;
        }
        else{
```

```
            if (pcb[current].state == STATE_RUNNABLE || pcb[current].state ==
STATE_RUNNING){
                pcb[current].timeCount = 0;
            }
            else
                current = 0;
        }
    }

    tmpStackTop = pcb[current].stackTop;
    pcb[current].stackTop = pcb[current].prevStackTop;
    tss.esp0 = (uint32_t)&(pcb[current].stackTop);
    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;
}
```

并且对`irqhandle()`函数做出一定的修改，和手册中一样，这里就不赘述了。

系统调用例程

syscallFork

`syscallFork()`要做的是在寻找一个空闲的`pcb`做为子进程的进程控制块，将父进程的资源复制给子进程。如果没有空闲`pcb`，则fork失败，父进程返回-1，成功则子进程返回0，父进程返回子进程`pid`

首先查找是否有空闲的进程，即状态为`STATE_DEAD`的进程，如果没有则父进程返回-1。

```
    int i;
    for (i = 0; i < MAX_PCB_NUM; i++)
    {
        if (pcb[i].state == STATE_DEAD)
            break;
    }
```

如果找到了，首先将内存进行复制

```
for (j=0; j<0x100000; j++) {
    *(uint8_t*)(j+ (i+1) *0x100000) =*(uint8_t*)(j+ (current+1) *0x100000);
}
```

再在`pcb`表中进行对`processtable`的复制

```
for (int j = 0; j < sizeof(ProcessTable); j++)
    *((uint8_t *)(&pcb[i]) + j) = *((uint8_t *)(&pcb[current]) + j);
```

设置栈顶指针，设置状态为`STATE_RUNABLE`，`timecount`和`sleeptime`都是0，`pid`为`i`，设置段寄存器的值，最后设置`eax`，子进程为0，父进程返回子进程`pid`，相关代码如下：

```
        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;

        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);

        pcb[i].regs.eax = 0;
        pcb[current].regs.eax = i;
```

syscallSleep

将当前的进程的`sleepTime`设置为传入的参数，将当前进程的状态设置为`STATE_BLOCKED`，然后利用`int $0x20`模拟时钟中断进行进程切换，注意参数存放在ecx中。

实现代码如下：

```
void syscallSleep(struct StackFrame *sf){
    int time = sf->ecx;
    if(time < 0)
        return;
    pcb[current].sleepTime = time;
    pcb[current].state = STATE_BLOCKED;
    asm volatile("int $0x20");
}
```

syscallExit

将当前进程的状态设置为`STATE_DEAD`，然后模拟时钟中断进行进程切换

代码如下：

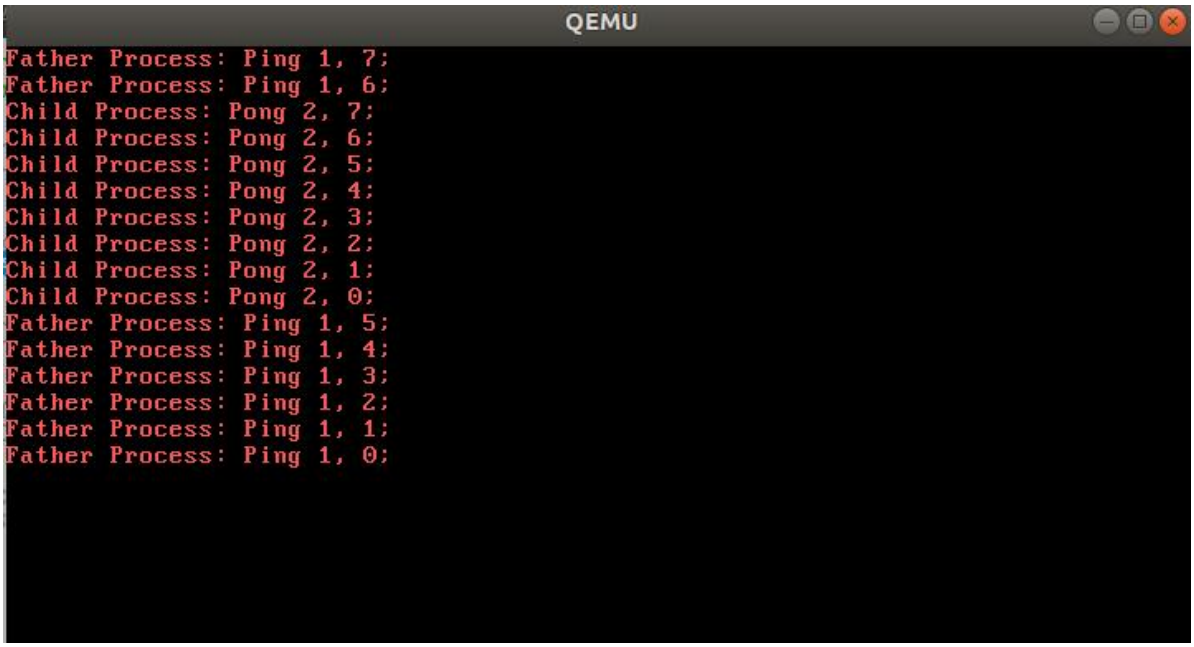
```
void syscallExit(struct StackFrame *sf){
    pcb[current].state = STATE_DEAD;
    pcb[current].timeCount = MAX_TIME_COUNT;
    asm volatile("int $0x20");
}
```

中断嵌套

可以用`enableInterrupt()`开启嵌套中断，再用`int $0x20`模拟时钟中断，如下：

```
        enableInterrupt();
        for (int j = 0; j < 0x100000; j++)
        {
            *(uint8_t *)(j + (i + 1) * 0x100000) = *(uint8_t *)(j + (current +
1) * 0x100000);
            asm volatile("int $0x20");
        }
        disableInterrupt();
```

实验效果



实验心得

通过代码模拟实现进程切换机制，了解进程切换的过程以及一些注意点。