操作系统 Lab3 实验报告

141220132@smail.nju.edu.cn 141220132 银琦

实验进度

完成了实验，实现了调度函数，实现了 fork，sleep，exit，循环输出 ping 和 pong。

实验结果截图：

实验过程及关键代码截图：

1. 初始化时钟中断

   将讲义中给的时钟中断初始化的代码放入新建的 timer.c 中，并放在 kernel 文件夹下。然后在中断向量表中添加时钟中断为 0x20，并在 do_irq.S 中添加相应的代码。

2. 初始化用户进程

   创建 PCB。新建了 pcb.c 和 pcb.h，在 pcb.h 中定义了 PCB 的结构体如下：

```
enum State {Runnable, Blocked, Running, Dead};

typedef struct PCB{
    TrapFrame *tf;
    int state;
    int time_count;
    int sleep_time;
    unsigned int pid;
    char name[32];
    uint8_t stack[4096];
    struct PCB* prev;
    struct PCB* next;
}PCB;
```

一开始按照讲义的结构定义了 PCB，但是使用数组处理较为麻烦，于是将结构改成了双向链表。定义了 PCB 的状态：就绪态、阻塞态、运行态和空闲状态。在 pcb.h 中借用了从网上找的 yzh 写好的结构，构造了对链表节点增删的函数，简化了操作。

建立了 pcb.c 文件，先写调度函数。Lab2 是通过 enter_user_space 进入看用户空间，lab3 则是从调度函数进入用户空间。首先初始化 pcb 表，设置每一个节点的状态、时间片、睡眠时间和 pid。我设置了 3 个链表，Ready 为就绪队列，Sleep 为阻塞队列，Free 为空闲队列，遍历一遍 PCB 数组，将每个节点加入相应的队列。然后将第 0 个节点初始化为用户进程如下：

```
pcb_table[0].tf = (void*)(pcb_table[0].stack + 4096 - sizeof(struct TrapFrame));
pcb_table[0].tf->ds = USEL(SEG_UDATA);
pcb_table[0].tf->es = USEL(SEG_UDATA);
pcb_table[0].tf->eip = elf->entry;
pcb_table[0].tf->cs = USEL(SEG_UCODE);
pcb_table[0].tf->eflags = 0x202;  //0x202
pcb_table[0].tf->esp = 0x200000;
pcb_table[0].tf->ss = USEL(SEG_UDATA);
pcb_table[0].state = Runnable;
pcb_table[0].time_count = 10;
```

然后开始写调度函数，设置 current 指针，指向当前运行的节点，判断如果 current 指向的是 IDLE 线程，当 Ready 非空时，就将 current 指向 Ready 队列的第一个节点，否则继续 IDLE 线程。判断时间片是否到期，若到期了就将当前节点的状态置为就绪态，从 Ready 队列删除并放入 Ready 队列的队尾，然后将 current 指向下一个节点，开始运行。调度函数如下：

```
void schedule() {
        if (current == &idle) {
                if (Ready != NULL) {
                        current = Ready;
                        current->time_count = 10;
                        current->state = Running;
                        tss.esp0 = (int)current->stack + 4096;
                }
                return;
        }
        if (current->time_count == 0) {
                PCB *p = current;
                if (p->state == Running)
                        p->state = Runnable;
                PCB *q = Ready;
                while(q->next != NULL)
                        q = q->next;
                if (p != q) {
                        list_del(p);
                        list_add_after(q, p);
                        while(q->prev != NULL)
                                q = q->prev;
                }
                Ready = q;
                current = Ready;
                current->state = Running;
                current->time_count = 10;
        }
        return;
}
```

最后修改一下 do_irq.S 文件，将 current 放入 esp 中，如下：

```
mov (current), %eax
mov (%eax), %esp
```

此时若 PCB 初始化没有问题，便进入了用户空间，可以将测试用例正确输出。

3. 系统调用

在 lib 中添加了三个系统调用相关的函数，这三个系统调用都属于 0x80 中断，在 irq_handle 中添加相关的系统调用情况，如下：

```
void do_syscall(struct TrapFrame *tf) {
        switch(tf->eax) {
                case 4: {
                        //int len = tf->edx;
                        //char *buf = (void *)tf->ecx;
                        putchar(tf->ecx);
                }break;
                case 1:{int sig = tf->ecx; myexit(sig);}break;
                case 2: {
//                      putchar('P');
                        myfork();
                }break;
                case 7:{ uint32_t time = tf->ecx; mysleep(time);}break;
                default:assert(0);
        }
}
```

我最先写的是 sleep，当检测到 sleep 函数后产生 80 中断，然后读出 tf->eax 的值，接着进入自己定义的 mysleep 函数，将当前节点的睡眠时间置为传入的参数，状态置为阻塞态。然后将这个节点从就绪队列里删除，放入睡眠队列，然后根据就绪队列的状态给 current 赋值。代码如下：

```
void mysleep(unsigned time) {
        PCB *p;
        current->sleep_time = time * HZ;
        current->state = Blocked;
        p = Sleep;
        if (current->next == NULL) {
                list_del(current);
                Ready = NULL;
        }
        else if (current->next != NULL) {
                Ready = current->next;
                list_del(current);
                current->next = NULL;
        }
        if (Sleep == NULL) {
                Sleep = current;
                Sleep->next = NULL;
        }
        else {
                while (p->next != NULL)
                        p = p->next;
                list_add_after(p,current);
        }
        if (Ready == NULL) {
                current = &idle;
        }
        else {
                current = Ready;
                current->time_count = 10;
                current->state = Running;
                tss.esp0 = (int)current->stack + 4096;
        }
}
```

每产生一个时钟中断，就将当前节点的 time_count--，每个睡眠队列中的 sleep_time--，并判断，如果有节点的 sleep_time 等于 0 了，就将其放入就绪队列的最末端。

```
void isBlocked() {
    PCB *p,*q,*s;
    p = Sleep;
    while (p != NULL) {
            p->sleep_time--;
            if (p->sleep_time <= 0) {
                    p->state = Runnable;
                    p->time_count = 10;
                    s = p;
                    if (s->next == NULL) {
                            p = p->next;
                            list_del(s);
                            Sleep = NULL;
                    }
                    else {
                            p = p->next;
                            list_del(s);
                            s->next = NULL;
                            Sleep = p;
                    }
                    q = Ready;
                    if (Ready == NULL)
                            Ready = s;
                    else {
                            while(q->next != NULL)
                                    q = q->next;
                            list_add_after(q,s);
                    }
            }
            else
                    p = p->next;
    }
void
irq_handle(struct TrapFrame *tf) {
    /*
     * 中断处理程序
     */
//      putchar('a');assert(0);
    current->tf = tf;
    switch(tf->irq) {
        case 0x80:do_syscall(tf);break;
        case 1000:break;
        case 1001:break;
        case 0x20:/*putchar('@');*/current->time_count--;isBlocked();
                break;
        case 0x2e:break;
        default:assert(0);
    }

    schedule();
}
```

接着写了 exit，这个函数相对比较简单，将当前节点的状态置为 Dead，将其放入空闲队列的最末端，并根据就绪队列的状态给 current 赋值。

```
void myexit(int sig) {
        PCB* p;
        p = Free;
        current->state = Dead;
        if (current->next == NULL) {
                list_del(current);
                while (p->next != NULL)
                        p = p->next;
                list_add_after(p,current);
                Ready = NULL;
                current = &idle;
        }
        else {
                Ready = current->next;
                list_del(current);
                while (p->next != NULL)
                        p = p->next;
                list_add_after(p,current);
                current = Ready;
                current->state = Running;
                current->time_count = 10;
                tss.esp0 = (int)current->stack + 4096;
        }
}
```

最后写了 fork 函数，一开始这个函数不太会写，在大神的帮助下慢慢理解了。再申请一块空间，给子进程用，将父进程拷贝进子进程申请的空间中，然后将子进程的 tf->eax 设置成 0，将父进程的 tf->eax 设置为子进程的 pid，最后将子进程加入就绪队列，即完成了 fork 函数。

```
void myfork() {
        gdt[6] = SEG(STA_X|STA_R, 0x400000, 0x200000, DPL_USER);
        gdt[7] = SEG(STA_W, 0x400000, 0x200000, DPL_USER);
        int i;
        for (i = 0; i < 0x200000; i++)
                *(char*)(0x400000+i) = *(char*)(0x200000+i);
        PCB *p = Free, *q = Ready;//, *s = Free->next;
        if (p == NULL)
                return;
        while(p->next != NULL)
                p = p->next;
        list_del(p);
        (*p) = (*current);
        p->tf = (void*)(((uint32_t)p - (uint32_t)current) + (uint32_t)current->tf);
        p->tf->ds = (USEL(7));
        p->tf->es = (USEL(7));
        p->tf->cs = (USEL(6));
        p->tf->ss = (USEL(7));
        p->state = Runnable;
        p->time_count = 10;
        p->tf->eax = 0;
        p->pid = 1;
        current->tf->eax = 1;
        if (Ready == NULL) Ready = p;
        else {
                while(q->next != NULL)
                        q = q->next;
                list_add_after(q,p);
        }
}
```

遇到的问题：

1. 一开始写调度函数时，始终没能进入用户空间，最后发现是没有将 current 指针放入 esp 中，再次出现了 lab2 中的错误，对栈结构处理错误。
2. 由于没有对 yzh 写好的结构中的节点增删函数进行改动，所以在实现双向链表的时候

容易出现 bug，后来发现写成双向循环链表要好操作一些。很多在表尾添加了节点，但是 next 不是 NULL 的情况导致了许多 bug。

3. 在写了 sleep 后始终没有进入时钟中断，仔细调试才发现，在 irq_handle 函数中，要保存寄存器状态，但是在 sleep 的时候，我把 current 改了，保存寄存器状态后一直讲用户进程的现场信息保存到 current 里的栈，然后中断返回的时候就返回用户进程。将保存寄存器的语句写在中断之前就没有问题了。
4. 在写 fork 的时候，一开始只能输出 ping 或者 pong，调试后发现是初始化的内存地址写大了，导致产生了错误，修改后能打出 ping 和 pong，但是顺序不太对，再次单步调试，仍然发现了 2 中的问题，最后一个节点的 next 不是 NULL，导致调度顺序混乱。

收获心得：

1. 要多熟悉一下 gdb 调试的指令，很多情况看代码看不出问题，但是一步步调试并输出一些值后就很容易看出问题。
2. 代码最好自己写，这次参考了 yzh 的代码，结果没有把他的函数理解透彻，导致了多次出现链表指针指向错误的问题。
3. 代码要写一部分测试一部分，如果写了一大段再测试就很难找到问题在哪里。
4. 写代码要细心，很多细节出错会导致结果大错，比如压栈顺序不同，比如地址的设置产生偏差等等。
5. 最后感谢大神的帮助。