实验七报告

【个人信息】

院系：数据科学与计算机学院　　专业、年级：17 级计算机科学与技术
学号：17341097　　姓名：廖永滨　　指导教师：凌应标

【实验目的】

1、 完善实验 6 中的二状态进程模型，实现五状态进程模型，从而使进程可以分工合作，并发运行。
2、 了解派生进程、结束进程、阻塞进程等过程中父、子进程之间的关系和分别进行的操作。
3、 理解原语的概念并实现进程控制原语 do_fork(), do_exit(),do_wait(), wakeup, blocked。

【实验要求】

1、 在实验五或更后的原型基础上，进化你的原型操作系统，原型保留原有特征的基础上，设计满足下列要求的新原型操作系统：
(1)实现控制的基本原语
(2)内核实现三系统调用，并在 c 库中封装相关的系统调用。
(3)编写一个 c 语言程序，实现多进程合作的应用程序。
2、 进程合作的应用程序可以在下面的基础上完成：由父进程生成一个字符串，交给子进程统计其中字母的个数，然后在父进程中输出这一统计结果。
3、 编译连接你编写的用户程序，产生一个 com 文件，放进程原型操作系统映像盘中。

【实验方案】

1. 实验工具：

Notepad++：编写程序时使用的编辑器；
Subline：可以以 16 进制的方式打开并编辑任意文件；
TAMS 汇编工具：可以将汇编代码编译成对应的二进制代码；
NAMS 汇编工具：可以将汇编代码编译成对应的二进制代码；
TCC 编译器：可以将 c 代码编译成对应的二进制代码；
TLINK 链接器：将多个.obj 文件链接成.com 文件
VmWare 虚拟机：创建裸机环境，生成虚拟磁盘

2. 实验基本操作方法：

a. 使用 TCC 编译命令：tcc -mt -c - o cfile.obj cfile.c >ccmsg.txt
b. TASM 汇编命令：tasm afile.asm afile.obj > amsg.txt
c. 链接命令：tlink /3 /t cfile.obj afile.obj , final.com
d. NASM 汇编指令：NASM afile.asm

3. 实验原理

中断机制基础知识，中断程序的编写规范。

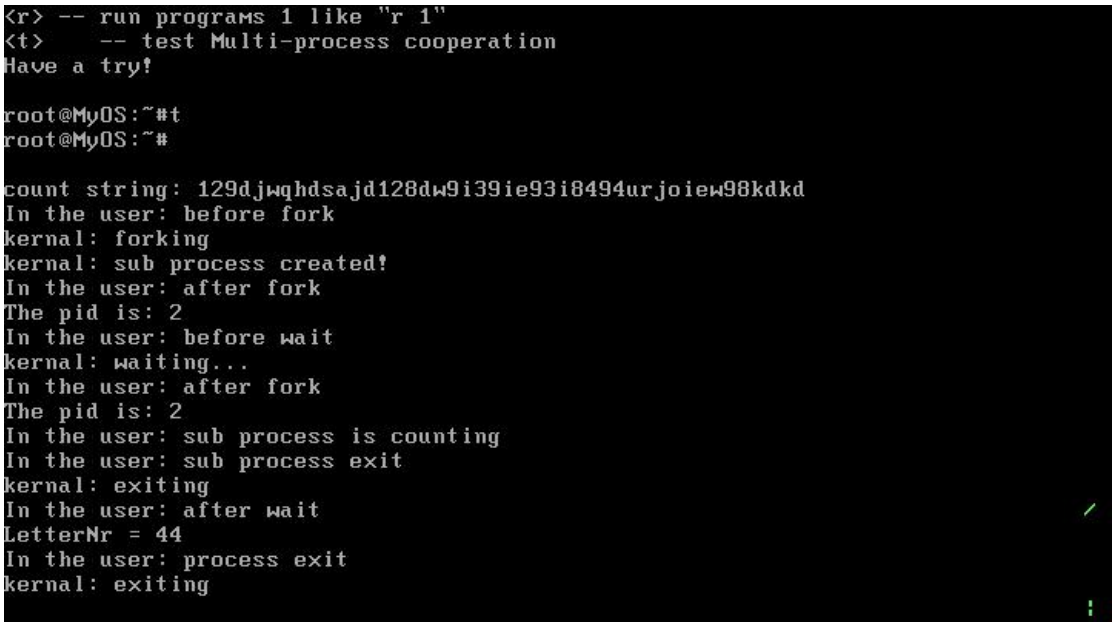
【实验过程】

1、实现的内容：

前一个原型中，操作系统可以用固定数量的进程解决多道程序技术。但是，这些进程模型还比较简单，进程之间互不往来，没有直接的合作。在实际的应用开发中，如果可以建立一组合作进程，并发运行，那么软件工程意义上更有优势。在这个项目中，我们完善进程模型，进程能够按需要产生子进程，一组进程分工合作，并发运行，各自完成一定的工作。合作进程在并发时，需要协调一些事件的时序，实现简单的同步，确保进程并发的情况正确完成使命。

这个实验有个难点在于设计合作程序，由于太过繁琐，而且我老是出 bug，我就直接搬来了别人合作子程序在自己的操作系统上跑，发现能完美运行，所以就没有改动。

实现效果：



输入指令 t

可以看到，各个进程间完成了合作行，严格按照 fork，wait 等原语来行动。

2、进程模型各模块说明：

（1）进程控制的基本操作：

（a）扩展 PCB 结构，增加必要的数据项

（b）进程创建 do_fork()原语，在 c 语言中用 fork()调用

（c）进程终止 do_exit()原语，在 c 语言中用 exit(int exit_value)调用

（d）进程等待子进程结束 do_wait()原语，在 c 语言中用 wait(&exit_value)调用

（e）进程唤醒 wakeup 原语（内核过程）

（f）进程唤醒 blocked 原语（内核过程）

（2） 进程创建：

进程的派生过程 do_fork()函数所执行的操作：先在 PCB 表查找空闲的表项，若没有找到，则派生失败。若找到空闲的 PCB 表项，则将父进程的 PCB 表复制给子进程，并将父进程的堆栈内容复制给子进程的堆栈，然后为子进程分配一个 PCBID，共享代码段和数据段，并且父子进程中对 do_fork()函数的返值分别设置。内核完成进程创建后，系统中增加了一个进程。

（3） do_fork 语句解析：

创建进程功能要指定一个系统调用号，在内核的系统调用总控程序中，增加一个分支，调用 do_fork() 完成子进程创建。系统调用的返回值获得方式：C 语言用 ax 传递，所以放在进程 PCB 的 ax 寄存器中。需要做到：1.寻找一个自由的 PCB 块，如果没有，创建失败，调用返回－1。2.以调用 fork()的当前进程为父进程，复制父进程的 PCB 内容到自由 PCB 中。3.产生一个唯一的 ID 作为子进程的 ID，存入至 PCB 的相应项中。4.为子进程分配新栈区，从父进程的栈区中复制整个栈的内容到子进程的操作系统原理实验栈区中。5.调整子进程的栈段和栈指针，子进程的父亲指针指向父进程。即 pcb_list[s].fPCB= pcb_list[CurrentPCBno];6.在父进程的调用返回 ax 中送子进程的 ID，子进程调用返回 ax 送 0。

（4）wait 语句解析：

在以前的原型操作系统中顺序执行用户程序，内存中不会同时有两个用户程操作系统原理实验序，所以 CPU 控制权交接问题简单，操作系统加载了一个用户到内存 中，然后将控制权交接给用户程序，用户程序执行完再将控制权交接回操作系统，一次性完成用户程序的执行过程。而在本次实验中采用时钟中断打断执行中的用户程序实现 CPU 在进程之间交替。简单起见，我们让两个用户的程序均匀地推进，就可以在每次时钟中断处理时，将 CPU 控制权从当前用户程序交接给另一个用户程序。

（5）exit 语句解析：

父进程如果想等待子进程结束后再处理子进程的后事，需要一个系统调用 wait()，进程被阻塞。而子进程终止时，调用 exit()，向父进程报告这一事件，可以传递一个字节的信息给父进程，并解除父进程的阻塞，并调用进程调度过程挑选另一个就绪进程接权。

3、代码解析（实现方法）

(1)MyOS.asm：

基本无变化，但是删掉了很多我觉得对实验 7 无用的中断。

(2)kliba.asm：

此处为中断的核心所在，在实验六的基础上，新增了很多过程，比如 fork，wait，exit 等等。

```
374  int_21h:
375  ⟶push bp
376  ⟶push ds
377  ⟶push es
378  ⟶
379  ⟶mov bx, cs
380  ⟶mov ds, bx
381  ⟶mov es, bx
382  ⟶
383  ⟶cmp ah, 4
384  ⟶je to_forking
385  ⟶cmp ah, 5
386  ⟶je to_waiting
387  ⟶cmp ah, 6
388  ⟶je to_exiting
389  ⟶jmp end21h
390
391  to_forking:
392  ⟶pop es
393  ⟶pop ds
394  ⟶pop bp
395  ⟶jmp forking
396
397  to_waiting:
398  ⟶pop es
399  ⟶pop ds
400  ⟶pop bp
401  ⟶jmp waiting
402
403  to_exiting:
404  ⟶pop es
405  ⟶pop ds
406  ⟶pop bp
407  ⟶jmp exiting
```

```
; 进程创建
forking:
    .386
    push ss
    push gs
    push fs
    push es
    push ds

    .8086
    push di
    push si
    push bp
    push sp
    push dx
    push cx
    push bx
    push ax

    mov ax,cs
    mov ds, ax
    mov es, ax

    call _save_PCB
    call near ptr _do_fork ; 调用 C 过程
    iret

waiting:
    .386
    push ss
    push gs
    push fs
    push es
    push ds

    .8086
    push di
    push si
    push bp
    push sp
    push dx
    push cx
    push bx
    push ax

    mov ax,cs
    mov ds, ax
    mov es, ax

    call _save_PCB
    call near ptr _do_wait ; 调用 C 过程
    iret
```

```
exiting:
    .386
    push ss
    push gs
    push fs
    push es
    push ds

    .8086
    push di
    push si
    push bp
    push sp
    push dx
    push cx
    push bx
    push ax

    mov ax,cs
    mov ds, ax
    mov es, ax

    call _save_PCB
    call near ptr _do_exit ; 调用 C 过程
    iret
```

(3)kernal.c ：

新增的进程相关的一系列函数。

```
129 int do_fork() {
130     int sub_ID;
131     print("kernal: forking\r\n");
132     sub_ID = createSubPCB();
133     if (sub_ID == -1) {
134         current_process_PCB_ptr->regs.ax = -1;
135         return -1;
136     }
137     sub_PCB = &PCB_LIST[sub_ID];
138     current_process_PCB_ptr->regs.ax = sub_ID;
139     sub_ss = sub_PCB->regs.ss;
140     f_ss = current_process_PCB_ptr->regs.ss;
141     stack_size = 0x100;
142     stackCopy();
143     PCB_Restore();
144 }
```

```
173 void blocked() {
174     current_process_PCB_ptr->status = PCB_BLOCKED;
175     schedule();
176     PCB_Restore();
177 }
178 void do_wait() {
179     print("kernal: waiting...\r\n");
180     blocked();
181 }
182
```

```
184 void wakeup() {
185     if (process_number == 1)
186         kernal_mode = 1;
187     schedule();
188     PCB_Restore();
189 }
190 void do_exit(int ss) {
191     print("kernal: exiting\r\n");
192     PCB_LIST[current_process_number].status = PCB_EXIT;
193     PCB_LIST[current_process_PCB_ptr->FID].status = PCB_READY;
194     PCB_LIST[current_process_PCB_ptr->FID].regs.ax = ss;
195     current_seg -= 0x1000;
196     process_number--;
197     wakeup();
198 }
199
```

其实这里改动是非常巨大的，因为我删掉了许多没用的东西，就连先前的 tab 代码补全也被我注释掉了（因为我简化了指令，补全功能太过鸡肋）

(4) user.c 等等 ：

由于实验要求我们设计一个能体现进程合作的子程序，但是我搞了很久都有 bug，就直接搬运的别人的代码，发现能完美契合我的操作系统，所以就没有进一步改动了。

【实验心得】

这次实验，最大的收获无异于对进程模型的进一步了解与合作方式的实现。

在实验六的基础之上，实验七的完成十分十分的困难，因为是出现了一堆乱七八糟的 bug，而且完全没有解决思路。

但是我也学到了很多 debug 的技巧，比如说将 print 语句插入到 fork 等各个函数中间，我就能很清晰地知道到底是哪个环节出了问题。

首先，我又遇到了退不回内核态的问题。也就是回不到风火轮的状态，风火轮在轮转完进程后就不再出现。一开始解决方法很牵强，就是每次回去都强制初始化，后面发现将不是实验要的做法，不是真正的进程模型，最后也不知道是怎

么的，把 iret 这些语句重新写了写就过了。。。。

接着，我发现 run_process 有问题，跑一半就死机了。这个问题困扰了我很久很久，最后发现是代码逻辑问题。在这两个问题解决的基础上，实验 7 基本上就完成了，子程序我是直接搬运的，所以就没有遇到什么问题。

总的来说，实验七也非常困难，我不得不参考别人代码，最后还出现了各种各样的 bug。由于我删除了实验 6 的基本所有实验 7 用不上的内容，所以内核很精简。

【代码清单】

```
Project7
    --操作系统
        --KERNAL.C
        --KLIBA.ASM
        --MYOS.ASM
    --工具包
        --DOSBox.exe（只是快捷方式）
        --mydoc.conf  (DOSBOX 的配置文件)
        --nasm.exe
        --nasmpath.bat
        --TASM.EXE
        --TCC.EXE
        --TLINK.EXT
    --软盘文件
        --实验 7.flp
    --引导程序
        --loading.asm
        --loading.com
    --用户程序
        --user.asm
        --user_lib.asm
        --user_lib.h
        --userc.c
        --user.com
        --userpro1.com
        --userpro2.com
        --userpro3.com
        --userpro4.com
```