
操作系统课程设计

项目文档和功能说明文档

1352834 安哲宏

1352918 刘旭东

1353010 薛梦迪



一、项目介绍

（一）项目简介

此项目是基于虚拟计算机Bochs所写的一个小型计算机操作系统，其中包含了进程管理与调用模块、输入输出处理模块，开关机功能，小游戏功能，计算器功能以及开关机动画和进程切换。

通过完成此项目，我们对计算机操作系统的理解更深了一层，知道了什么是实模式和保护模式；通过编写修改此操作系统的内核，我们也更加理解了操作系统内核运转的方式；通过实际操控、添加、修改进程和进程调度算法，我们也更加理解了操作系统中对于进程的管理。

（二）开发环境

- 1) 虚拟机 VMware Fusion 7
- 2) 操作系统 ubuntu 14.04
- 3) 模拟器 Bochs 2.4.5
- 4) 编译器 gcc + Nasm

（三）参考书籍

- 1) 《一个操作系统的实现》于渊著 电子工业出版社
- 2) 《30天自制操作系统》川合秀实著 人民邮电出版社
- 3) 助教的开发环境搭建pdf

二、项目分工与安排

（一）项目分工

我们根据项目的主要模块做了如下分工：

- 刘旭东主要负责了项目整体的初始化，将进程调度算法改为多级调度，并写了一个用户级日历小应用。
- 安哲宏主要创建了开关机动画，对内部进程进行了调整，增加了颜色，美化了整个系统的界面。
- 薛梦迪主要负责写了一个计算器应用，修改了五子棋应用，并负责了小组的文档、ppt等工作。

（二）项目时间安排

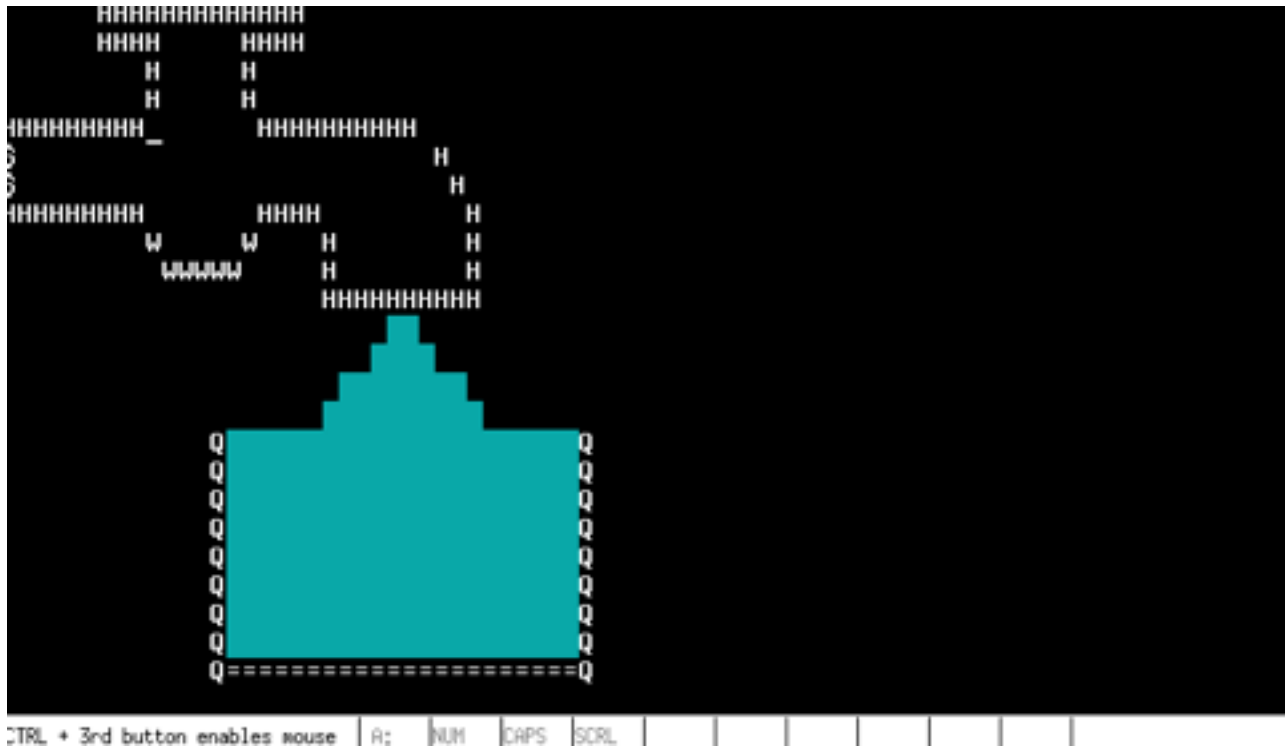
首先，通过助教给的开发环境教程，我们在VMware虚拟机中装入了ubuntu14.04，并且配置了gcc、nasm等编译必须的工具，接下来我们按照书中的方式安装了bochs并运行出第一个hello world。

接下来，我们借鉴了《一个操作系统的实现》中的Tinux源码编写了一个简单而小巧的操作系统，并对其不完善之处进行完善：例如进程调度算法的修改、进程状态的切换、添加开关机动画、添加系统级应用等。

最后，我们进行了代码的整合与测试，并编写了文档。

三、项目截图

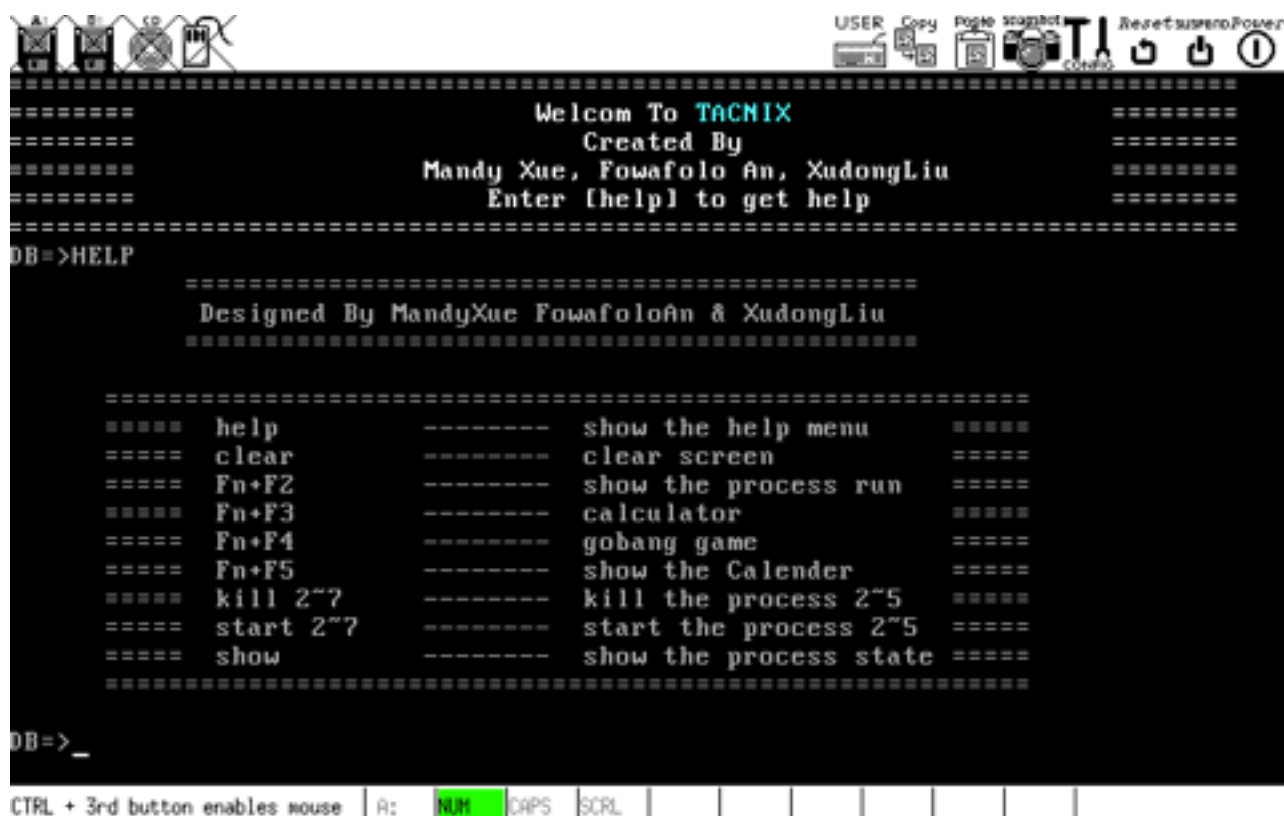
(一) 开机动画



(二) 桌面



(三) help



```
=====
                        Welcom To TACNIX
                        Created By
Mandy Xue, Fowafolo An, XudongLiu
                        Enter [help] to get help
=====
DB=>HELP
=====
      Designed By MandyXue FowafoloAn & XudongLiu
=====

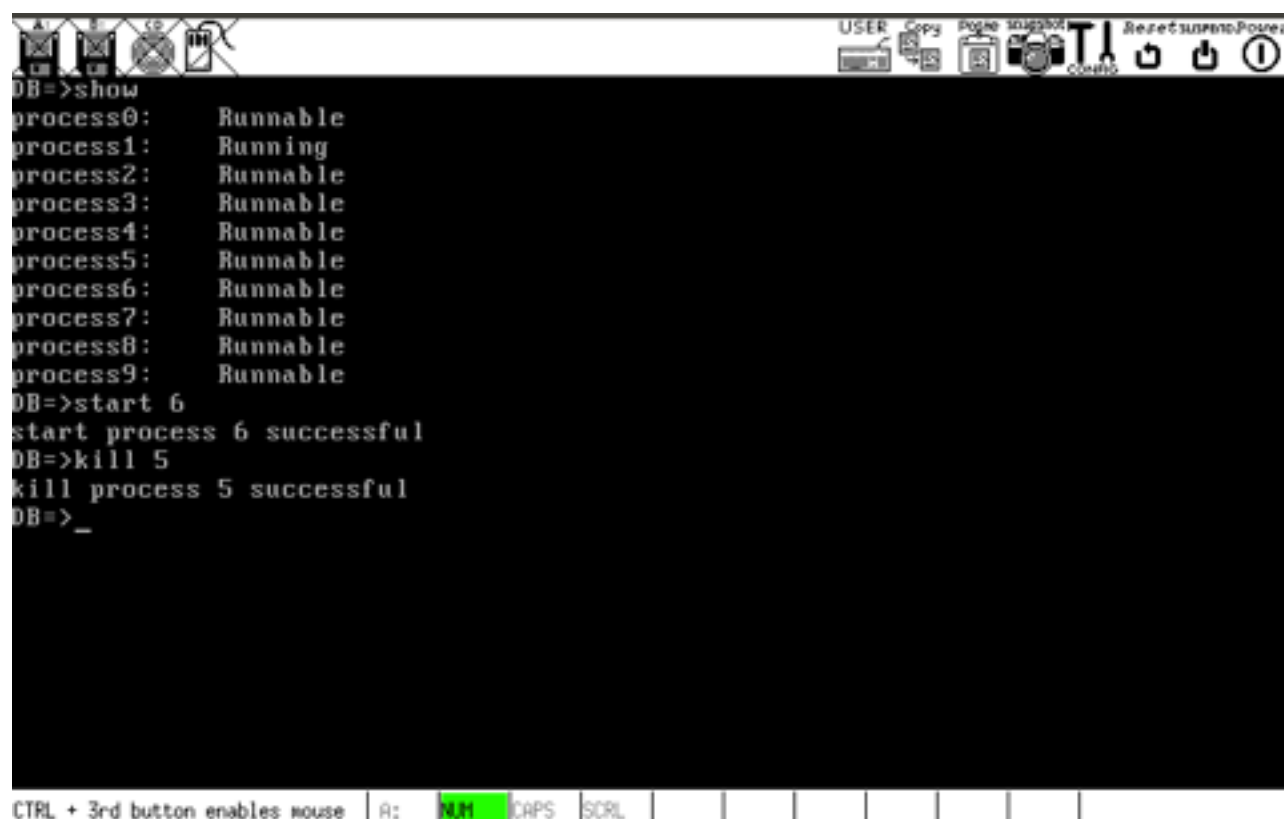
=====
===== help      ----- show the help menu =====
===== clear      ----- clear screen =====
===== Fn+F2      ----- show the process run =====
===== Fn+F3      ----- calculator =====
===== Fn+F4      ----- gobang game =====
===== Fn+F5      ----- show the Calender =====
===== kill 2~7    ----- kill the process 2~5 =====
===== start 2~7   ----- start the process 2~5 =====
===== show       ----- show the process state =====
=====

DB=>_

CTRL + 3rd button enables mouse | A: NUM CAPS SCRL | | | | | | | | | |
```

(四) 进程管理

1、状态管理



```
DB=>show
process0:    Runnable
process1:    Running
process2:    Runnable
process3:    Runnable
process4:    Runnable
process5:    Runnable
process6:    Runnable
process7:    Runnable
process8:    Runnable
process9:    Runnable
DB=>start 6
start process 6 successful
DB=>kill 5
kill process 5 successful
DB=>_

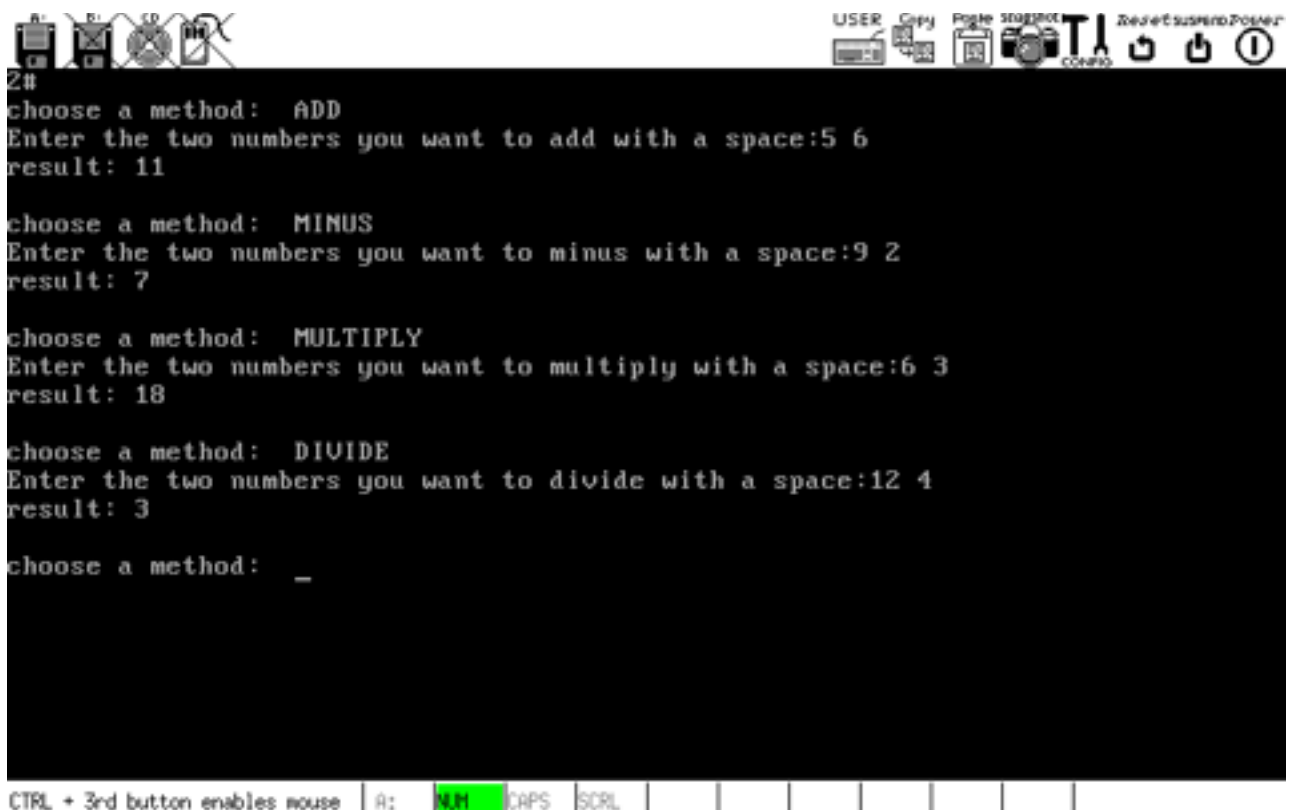
CTRL + 3rd button enables mouse | A: NUM CAPS SCRL | | | | | | | | | |
```

2、进程调度

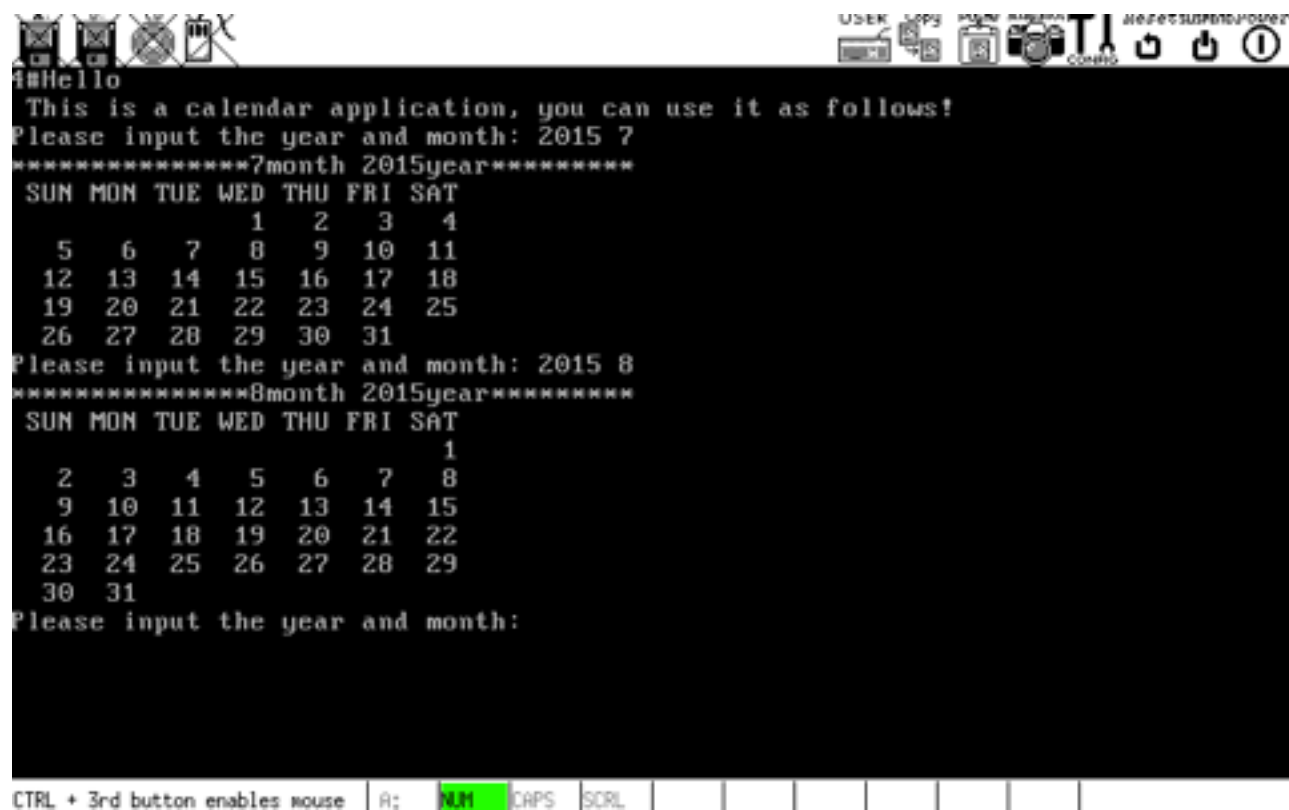


(五) 用户级应用

1、应用一：计算器

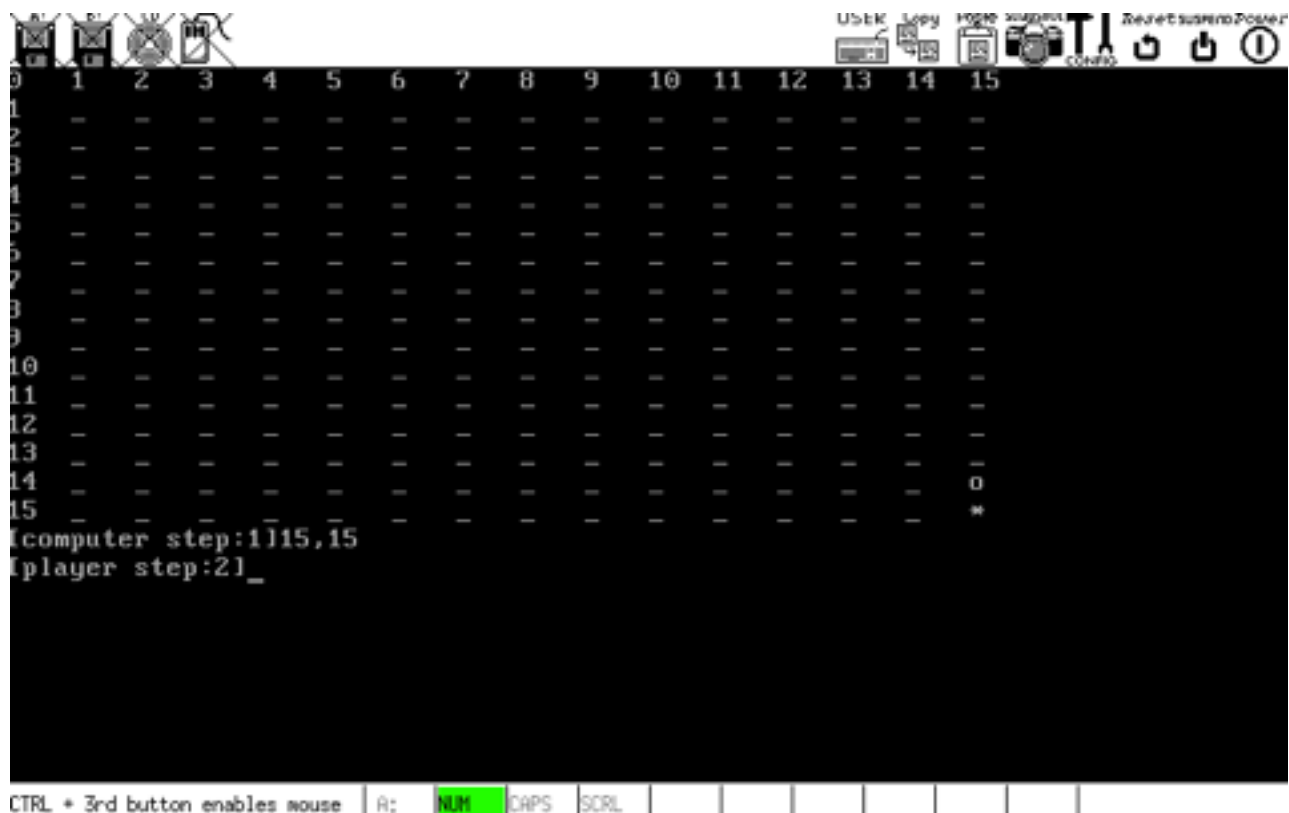


2、应用二：日历



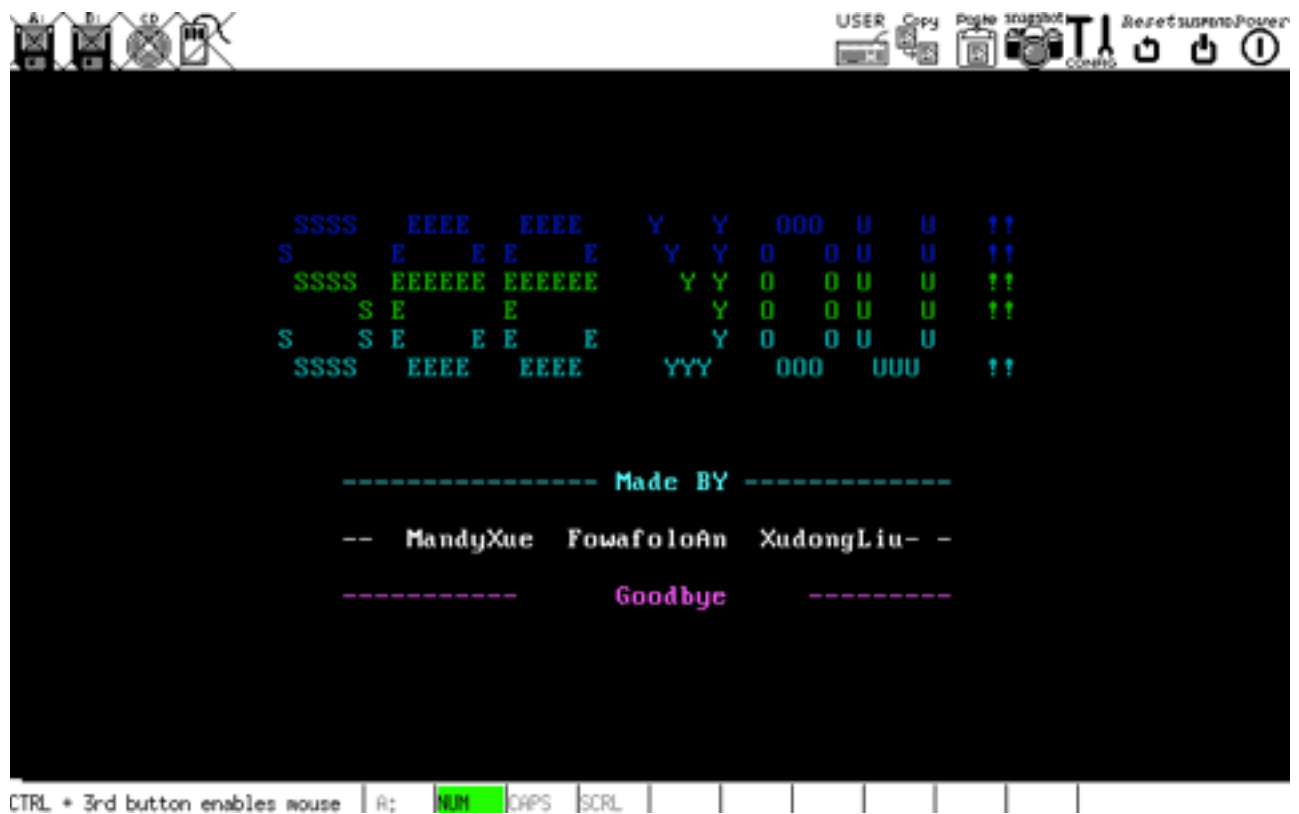
```
##Hello
This is a calendar application, you can use it as follows!
Please input the year and month: 2015 7
*****7month 2015year*****
SUN MON TUE WED THU FRI SAT
      1  2  3  4
  5  6  7  8  9 10 11
 12 13 14 15 16 17 18
 19 20 21 22 23 24 25
 26 27 28 29 30 31
Please input the year and month: 2015 8
*****8month 2015year*****
SUN MON TUE WED THU FRI SAT
              1
  2  3  4  5  6  7  8
  9 10 11 12 13 14 15
 16 17 18 19 20 21 22
 23 24 25 26 27 28 29
 30 31
Please input the year and month:
```

3、游戏：五子棋



```
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
1  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
2  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
3  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
4  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
5  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
6  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
7  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
8  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
9  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
10 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
11 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
12 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
13 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
14 -  -  -  -  -  -  -  -  -  -  -  -  -  0
15 -  -  -  -  -  -  -  -  -  -  -  -  -  *
[computer step:1115,15
[player step:21_
```

(六) 关机动画



四、代码说明

(一) 开机、关机动画的实现

在这部分中，我们的思路主要是希望用一个水龙头滴水的动画来体现我们开机加载的过程，加载成功之后即进入我们的桌面。具体实现时，我们调用了延迟函数`milli_delay()`和清屏函数`clearScreen()`来实现动画的帧数控制，其中`milli_delay`中参数越大，间隔时间越长。

以下是开机动画代码的具体实现：

```
/*=====
|           animation
|=====*/

void DisplayAnimation()//开机动画
{
    //////////////////////////////////////
    clearScreen();
    disp_str("          HHHHHHHHHHHH\n");
    disp_str("          HHHH      HHHH\n");
    disp_str("           H      H\n");
    disp_str("           H      H\n");
    disp_str("HHHHHHHHH      HHHHHHHHHH\n");
    disp_str("$              H\n");
    disp_str("$              H\n");
    disp_str("HHHHHHHHH      HHHH      H\n");
    disp_str("           W      W      H      H\n");
    disp_str("           WWWW      H      H\n");
    disp_str("           HHHHHHHHHH\n");
    disp_str("           ");
    disp_color_str("AA\n",0x33);
    disp_str("           ");
    disp_color_str("AAAA\n",0x33);
    disp_str("           ");
    disp_color_str("AAAAAAA\n",0x33);
    disp_str("           ");
    disp_color_str("AAAAAAA\n",0x33);

    disp_str("           Q              Q\n");
    disp_str("           Q              Q\n");
    disp_str("           Q              Q\n");
    disp_str("           Q              Q\n");
    disp_str("           Q              Q\n");
    disp_str("           Q              Q\n");
    disp_str("           Q");
    disp_color_str("AAAAAAAAAAAAAAAAAAAA",0x33);
    disp_str("Q\n");
    disp_str("           Q");
    disp_color_str("AAAAAAAAAAAAAAAAAAAA",0x33);
    disp_str("Q\n");
    disp_str("           Q-----Q\n");
    milli_delay(1);
}
```

延迟函数的具体实现：

```
/*=====
milli_delay
=====*/
PUBLIC void milli_delay(int milli_sec)
{
    int t = get_ticks();

    /*while(((get_ticks() - t) * 1000 / HZ) < milli_sec) {}*/

    int i = 0;

    while(i < milli_sec * 4000000)
    {
        i++;
    }

    return;
}
```

关于关机动画部分，我们打印出了彩色的SEE YOU来告知用户已成功关闭系统，并且这时用户不能再进行操作。

以下是关机动画代码的具体实现：

```
/*=====
display goodbye
=====*/

void displayGoodBye()
{
    clearScreen();
    disp_str("\n\n\n\n\n");
    disp_color_str("
SSSS  EEEE  EEEE  Y Y  000 U U  || \n",0x1);
    disp_color_str("
S   E   E   E   Y Y  0   O U  || \n",0x1);
    disp_color_str("
SSSS  EEEEE  EEEEE  Y Y  0   O U  || \n",0x2);
    disp_color_str("
   S E   E   Y   O   O U  || \n",0x2);
    disp_color_str("
S   S E   E   E   Y   O   O U  || \n",0x3);
    disp_color_str("
SSSS  EEEE  EEEE  YYY  000  UU  || \n",0x3);
    disp_str("\n\n\n");
    disp_color_str("
----- Made BY ----- \n\n",0x8);
    disp_color_str("
-- MandyXue FowafoloAn XudongLiu- - \n\n",0xF);
    disp_color_str("
----- Goodbye ----- \n\n",0xD);
}
```

而动画中的颜色，我们是在printf.c中定义了printf函数用来打印出字符串，但是我们发现这时候不能打印出彩色的文字，于是我们写了一个可以打印彩色文字的c_printf函数，同时修改了汇编的部分，如右图：

```
int printf(const char *fmt, ...)
{
    int i;
    char buf[256];

    va_list arg = (va_list)((char*)&fmt + 4);
    i = vsprintf(buf, fmt, arg);
    write(buf, i);

    return i;
}

int c_printf(int color, const char *fmt, ...)
{
    int i;
    char buf[256];

    va_list arg = (va_list)((char*)&fmt + 4);
    i = vsprintf(buf, fmt, arg);
    c_write(buf, i, color);

    return i;
}
```

汇编部分修改如下：

```
; =====  
;                                     void write(char* buf, int len);  
; =====  
write:  
    mov     eax, _NR_write  
    mov     ebx, [esp + 4]  
    mov     ecx, [esp + 8]  
    int     INT_VECTOR_SYS_CALL  
    ret  
  
; =====  
;                                     void c_write(char* buf, int len, int color);  
; =====  
c_write:  
    mov     eax, _NR_c_write  
    mov     ebx, [esp + 4]  
    mov     ecx, [esp + 8]  
    mov     edx, [esp + 12]  
  
    int     INT_VECTOR_SYS_CALL  
    ret  
  
_NR_get_ticks      equ     0  
_NR_write          equ     1  
_NR_c_write        equ     2
```

可惜的是，经过这样的更改，我们发现颜色还是不能正常的显示出来，由于时间有限，因为没有汇编的基础，我们没有继续研究汇编部分。而经过研究，我们了解到此系统中有内建函数 `disp_color_str("",0x)`，可以打印出一个用十六进制数编码的颜色，其中代码部分可以参照上面的实现。到此我们就完成了界面的优化。

（二）主进程部分的实现

1、help

帮助功能主要是为了用户能更方便地了解系统的功能，帮助的实现主要用了 `printf.c` 中的 `printf()` 函数打印出必要的帮助信息。

2、clear

清屏功能是为了避免主进程的页面打印不下的状况。当用户执行清屏命令时，系统会自动清空屏幕上所有字符并将光标移动到左上角。

清屏的代码实现部分主要如右：

```
/*=====*/  
/*                                     clearScreen                                     */  
/*=====*/  
void clearScreen()  
{  
    int i;  
    disp_pos=0;  
    for(i=0;i<80*25;i++)  
    {  
        disp_str(" ");  
    }  
    disp_pos=0;  
}
```

3、主进程的切换

主进程的切换主要通过终端函数Terminal()来实现，而Terminal()主要通过dealWithCommand()和终端TTY来实现。

主进程的TTY即为tty_table。此时，先打印出“DB=>”，然后用openStartScanf(p_tty)标志键盘开始，并用while循环读取输入的内容直至回车。读取好内容后，再通过dealWithCommand(p_tty->str)处理读取进来的指令。

Terminal()代码实现如下：

```
/*=====*/
/*                          Terminal                          */
/*=====*/
void Terminal()
{
    TTY *p_tty=tty_table;
    p_tty->startScanf=0;
    while(1)
    {
        printf("DB=>");
        //printf("<Ticks:X>", get_ticks());
        openStartScanf(p_tty);
        while (p_tty->startScanf) ;
        dealWithCommand(p_tty->str);
    }
}
```

其中dealWithCommand()部分采用了循环判断对输入的指令进行解析，clear()、help()、show()、displayGoodBye()等函数可以直接解析出来，而kill和start需要使用readOneStringAddOneNumber(command,str,&number)来将command中的指令和进程标识分别解析出来。

dealWithCommand()实现如下：

```
/*=====*/
/*                          dealWithCommand                  */
/*=====*/
void dealWithCommand(char* command)
{
    while(1)
    {
        strlwr(command);
        if (strcmp(command,"clear")==0)
        {
            =
        }
        if (strcmp(command,"help")==0)
        {
            =
        }
        if (strcmp(command,"show")==0)
        {
            =
        }
        if (strcmp(command,"goodbye")==0)
        {
            =
        }
        char str[10] = {'\0','\0','\0','\0','\0','\0','\0','\0','\0','\0'};
        int number;
        readOneStringAndOneNumber(command,str,& number);
        if (strcmp(str,"kill")==0)
        {
            =
        }
        if (strcmp(str,"start")==0)
        {
            =
        }
        printf("can not find this command\n");
    }
}
```

(四) 进程管理的实现

1、状态管理和转换的实现

进程管理中的状态管理主要通过start和kill指令实现，调用方法和主进程中其他指令的调用方式相同，唯一多出一个步骤即是拆分指令和进程标识。用户输入“指令 + 进程标识”（例如“kill 5”），这时系统将用readOneStringAndOneNumber(char* command,char* str,int* number)来将指令拆分，具体实现方式如下：

```
/*=====
readOneStringAndOneNumber
=====*/

void readOneStringAndOneNumber(char* command,char* str,int* number)
{
    int i;
    int j=0;
    for (i=0; i<strlen(command); i++)
    {
        if (command[i]!=' ') break;
    }
    for (; i<strlen(command); i++)
    {
        if (command[i]==' ') break;
        str[j]=command[i];
        j++;
    }
    for (; i<strlen(command); i++)
    {
        if (command[i]!=' ') break;
    }

    *number=0;
    for (; i<strlen(command) && '0'<=command[i] && command[i]<='9'; i++)
    {
        *number=*number*10+(int) command[i]-'0';
    }
}
```

这时，主进程会判断输入的指令能否执行并进行相应的处理，在dealWithCommand()中的实现如下：

```
if (strcmp(str,"kill")==0)
{
    if (number<0 || number>NR_TASKS+NR_PROCS)
    {
        printf("No found this process!!");
    }
    else if (number==0 || number==8)
    {
        printf("You do not have sufficient privileges\n");
    }
    else if (2<=number && number <=7)
    {
        proc_table[number].state=kREADY;
        printf("kill process %d successful\n",number);
    }
    return ;
}
```

2、进程调度算法的修改——三级调度算法的实现

本项目将源码中的优先级调度改成了多级队列的反馈调度，采用了三级进程队列，其中第一级队列采用了FIFO调度算法，第二级和第三级则都采用了优先级调度，第一级队列每个进程执行的时间片为2，第二级队列每个进程执行的时间片为3，第三级队列每个进程的时间片和优先级保持一致。

首先，在global.h中声明这三个队列：

```
EXTERN PROCESS*   p_proc_ready;
EXTERN PROCESS*   firstQueue[20];
EXTERN PROCESS*   secondQueue[20];
EXTERN PROCESS*   thirdQueue[20];
```

具体调度算法修改在proc.c中实现：

```
PUBLIC void schedule()
{
    PROCESS*   p;
    int         greatest_priority=0;
    int         i;

    while (!greatest_priority)
    {
        if (firstlen-firstHead>0)
        {
            p_proc_ready=firstQueue[firstHead]; //第一个队列按照先到先得
            greatest_priority=p_proc_ready->ticks;
            break;
        }
        else if (secondlen+firstlen - firstHead >0)
        {
            for (i=0; i<secondlen; i++) //the second queue use the priority algorithm
            {
                p=secondQueue[i];
                if (p->state!=kRUNNABLE || p->ticks==0) continue;
                if (p->ticks > greatest_priority)
                {
                    greatest_priority = p->ticks;
                    p_proc_ready = p;
                }
            }
        }
        else{
            for (i=0; i<thirdlen; i++) //the third queue still use the priority algorithm
            {
                p=thirdQueue[i];
                if (p->state!=kRUNNABLE || p->ticks==0) continue;
                if (p->ticks > greatest_priority)
                {
                    greatest_priority = p->ticks;
                    p_proc_ready = p;
                }
            }
        }
        if (!greatest_priority) initializeAllPro();
    }
    p_proc_ready->state=kRUNNING;
}
```


接下来，在clock.c中对队列进行了降级操作：

```
/*===== clock_handler =====*/
/*===== clock_handler =====*/
PUBLIC void clock_handler(int irq)
{
    //disp_str("#");
    ticks++;
    p_proc_ready->ticks--;

    if (k_reenter != 0) {
        //disp_str("!");
        return;
    }

    if (p_proc_ready->ticks > 0) {
        return;
    }

    if (p_proc_ready->whichQueue==1) //如果是第一个队列的，降到第二个队列
    {
        p_proc_ready->whichQueue=2;
        p_proc_ready->ticks=2;
        secondQueue[secondLen]=p_proc_ready;
        secondLen++;
        firstHead++;
    }
    else if(p_proc_ready->whichQueue ==2) //否则是第二个队列的
    {
        p_proc_ready->whichQueue=3;
        p_proc_ready->ticks=3;
        thirdQueue[thirdLen] = p_proc_ready;
        thirdLen++;
        firstHead++;
    }
    else
    {
    }

    p_proc_ready->state=kRUNNABLE;

    schedule();
}
```

最后，在main.c中对队列进行了初始化：

```
TASK*      p_task;
PROCESS*   p_proc      = proc_table;
char*      p_task_stack = task_stack + STACK_SIZE_TOTAL;
t_16       selector_ldt = SELECTOR_LDT_FIRST;
int         i;
t_8         privilege;
t_8         rpl;
int         eflags;
for(i=0;i<NR_TASKS+NR_PROCS;i++){
}

//修改这里的优先级和ticks
proc_table[0].priority = 15;

//对优先队列初始化
thirdLen=0;
firstLen=firstHead=secondLen=0;
for (i=0; i<NR_TASKS+NR_PROCS;i++)
{
    addToQueue(proc_table+i);
}
```

```
//指定控制台
proc_table[1].nr_tty = 0;
proc_table[2].nr_tty = 1;
proc_table[3].nr_tty = 1;
proc_table[4].nr_tty = 1;
proc_table[5].nr_tty = 1;
proc_table[6].nr_tty = 2;
proc_table[7].nr_tty = 3;
proc_table[8].nr_tty = 4;
proc_table[9].nr_tty = 5;

k_reenter = 0;
ticks = 0;
```

运行起来时可以看到，在相同的时间内，不同的进程由于优先级的不同运行的次数不同，大体的运行次数和他们的优先级相同。

这样就完成了进程算法的优化和修改，采用了更高级的进程调度算法，使操作系统的运行调度更加流畅。

3、进程切换的实现

通过help可以看到，其中有几个Fn+Fx的指令，即为进程之间切换的实现。

首先，在main.c中写好进程内执行的代码，例如：

```
/*-----*
|         |         |         |         |         |         |         |         |
|-----|-----|-----|-----|-----|-----|-----|-----|
*-----*/
void appone()
{
    printf("Hello \n This is a calendar application, you can use it as follows!\n");
    calendar();
    printf("byebye\n");
}
```

在global.c中添加task_table:

```
PUBLIC TASK task_table[NR_TASKS] = {{task_tty, STACK_SIZE_TTY, "tty"}};
PUBLIC TASK user_proc_table[NR_PROCS] = { {Terminal, STACK_SIZE_TERMINAL, "Terminal"},
{TestB, STACK_SIZE_TESTB, "TestB"},
{TestC, STACK_SIZE_TESTC, "TestC"},
{TestD, STACK_SIZE_TESTD, "TestD"},
{TestE, STACK_SIZE_TESTE, "TestE"},
{calculator, STACK_SIZE_TESTE, "calculator"},
{goBangGameStart, STACK_SIZE_GOBANGGAMESTART, "GoBangGame"},
{appone, STACK_SIZE_TESTE, "appone"},
{apptwo, STACK_SIZE_TESTE, "apptwo"}};
```


在proc.h中添加STACK_SIZE:

```
#define STACK_SIZE_TOTAL (STACK_SIZE_TTY + \
    STACK_SIZE_TERMINAL + \
    STACK_SIZE_TESTB + \
    STACK_SIZE_TESTC + \
    STACK_SIZE_TESTD + \
    STACK_SIZE_TESTE + \
    STACK_SIZE_TESTE + \
    STACK_SIZE_TESTE + \
    STACK_SIZE_TESTE + \
    STACK_SIZE_GOBANGGAMESTART)
```

在proto.h中添加函数声明:

```
/* main.c */
PUBLIC void Terminal();
PUBLIC void TestB();
PUBLIC void TestC();
PUBLIC void TestD();
PUBLIC void TestE();
PUBLIC void clearScreen();
PUBLIC void goBangGameStart();
PUBLIC void calculator();
PUBLIC void appone();
PUBLIC void apptwo();
```

在main.c中给新进程赋予优先级并制定控制台:

```
//修改这里的优先级和ticks
proc_table[0].priority = 15;
proc_table[1].priority = 5;
proc_table[2].priority = 5;
proc_table[3].priority = 5;
proc_table[4].priority = 2;
proc_table[5].priority = 10;
proc_table[6].priority = 20;
proc_table[7].priority = 8;
proc_table[8].priority = 8;
proc_table[9].priority = 8;
```

```
//指定控制台
proc_table[1].nr_tty = 0;
proc_table[2].nr_tty = 1;
proc_table[3].nr_tty = 1;
proc_table[4].nr_tty = 1;
proc_table[5].nr_tty = 1;
proc_table[6].nr_tty = 2;
proc_table[7].nr_tty = 3;
proc_table[8].nr_tty = 4;
proc_table[9].nr_tty = 5;
```

在global.h中修改gdt_ptr和idt_ptr数量（总进程数）：

```
EXTERN t_8 gdt_ptr[9]; // 0~15:Limit 16~47:Base
EXTERN t_8 idt_ptr[9]; // 0~15:Limit 16~47:Base
```

在const.h中修改总控制台数量NR_CONSOLES:

```
/* TTY */
#define NR_CONSOLES 6 /* consoles */
```

最后，重新make image并运行即可。

（五）用户级应用的实现

1、应用一：计算器

计算器的主要思想为：让用户输入指令后采取相应的判断，让用户再次输入加（减/乘/除）数和被加（减/乘/除）数，最后返回结果。

给此进程赋予相应的TTY:

```
TTY *calculatorTty=tty_table+2;
```

然后实现循环函数dealWithCal(char *command):

```
void dealWithCal(char* command)
{
    strlwr(command);
    if (strcmp(command,"add")==0)
    {
        // clearScreen();
        // sys_clear(tty_table);
        disp_str("Enter the two numbers you want to ");
        disp_color_str("add",0xB);
        disp_str("with a space\n");
        // printf("Enter the two numbers you want to add with a space:");
        //scanf("%d%d",&x,&y);
        openStartScanf(calculatorTty);
        while (calculatorTty->startScanf) ;
        int x,y;
        readTwoNumbers(&x,&y);
        int result = add_fun(x,y);
        printf("result: %d\n", result);
        return ;
    }
    if (strcmp(command,"minus")==0)
    {
        //
    }
    if (strcmp(command,"multiply")==0)
    {
        //
    }
    if (strcmp(command,"divide")==0)
    {
        //
    }

    char str[10] = {'\0','\0','\0','\0','\0','\0','\0','\0','\0','\0'};
    int number;
    readOneStringAndOneNumber(command,str,& number);

    printf("can not find this command\n");
}
```

最后实现calculator()进行循环处理：

```
void calculator()
{
    printf("Hello\n");
    printf("This is a calculator application, you can enter\n");
    printf("add/minus/multiply/divide to calculate\n");
    TTY *p_tty2=tty_table+2;
    p_tty2->startScanf=0;
    while(1){
        printf("\nchoose a method: ");
        openStartScanf(p_tty2);
        while (p_tty2->startScanf) ;
        dealWithCal(p_tty2->str);
    }
}
```

2、应用二：日历

日历的主要思想为读取两个数字，对两个数字进行处理打印出相应年月日的日历，并且日历中也考虑了闰年。

处理两个数字的函数为readTwoNumberInCalendar(int* x, int* y)，实现如下：

```
void readTwoNumberInCalendar(int* x,int* y)
{
    int i=0;
    *x=0;
    *y=0;
    for (i=0; i<calendarTty->len && calendarTty->str[i]!=' '; i++);
    for (; i<calendarTty->len && calendarTty->str[i]!=' ' && calendarTty->str[i]!='\n'; i++)
    {
        *x=(*x)*10+(int) calendarTty->str[i]-48;
    }
    for (i; i<calendarTty->len && calendarTty->str[i]!=' '; i++);
    for (; i<calendarTty->len && calendarTty->str[i]!=' ' && calendarTty->str[i]!='\n'; i++)
    {
        *y=(*y)*10+(int) calendarTty->str[i]-48;
    }
}
```

然后编写打印函数print(int day, int tian)，最后通过rili(int year, int month)打印出所需年月的日历。

3、游戏：五子棋

五子棋的主要思想为：首先让用户通过selectPlayerOrder()选择先下棋的人，然后刷新整个界面，打印出现在的棋盘状态并让用户选择下棋的点，用户输入下棋点后，计算机通过算法getPossible()来计算接下来需要下棋的点，选择其中一点然后返回用户继续输入下棋点，直到一方赢得游戏。

四、项目优点和不足之处

（一）优点

此项目运行顺畅，思路较为清晰，并且在源代码的基础上实现了许多附加的功能，例如开关机动画和多个用户级应用，还修改了内核级的程序，例如进程调度算法和进程和控制台的添加。

（二）缺点

项目中有些算法还有优化的空间，例如计算器的实现较为简单，若能支持表达式输入则更好。并且，由于难度较大，此操作系统也没有实现GUI，有些程序的执行也有欠佳之处，希望以后能有所改进。