

# 同济大学计算机系

## 操作系统课程设计报告



班级 计算机三班

学号 1752669

姓名 杨燕

完成日期 2020/5/15

指导老师 邓蓉

## 目录

1 实验目的.....	3
2 实验设备及工具.....	3
3 基础知识.....	3
1: 字符终端的输入输出.....	3
2: 界面显示.....	4
3: 键盘硬件和中断处理程序.....	4
4: Unix V6++的初始化过程.....	5
4 实验内容.....	5
1: 明确自己的目标.....	5
2: 查看 Unix V6++的源码, 理解单用户系统下用户是如何打开并绑定 tty 终端的。...5	
3: TTy 终端的输入数据流程.....	6
4: 终端显示 (CRT), 查看终端数据是怎么显示在屏幕上的。.....	7
5: 改写代码, 实现两个终端.....	8
1: 修改 CRT 的代码: .....8	
2: 修改 TTy 的代码.....9	
3: 修改 CharDevice 的代码: .....9	
4 修改 Keyboard 代码.....10	
5 修改 shell.c.....10	
6 关于/dev/tty2 文件的创建问题。.....	11
5 运行结果.....	12
6 总结反思.....	13

## 1 实验目的

修改 UNIX V6++ 内核代码，使之能够支持多用户同时使用。在深入理解操作系统多用户概念和多用户概念和多用户系统实现细节的同时，培养剖析大型软件、设计系统程序的能力。

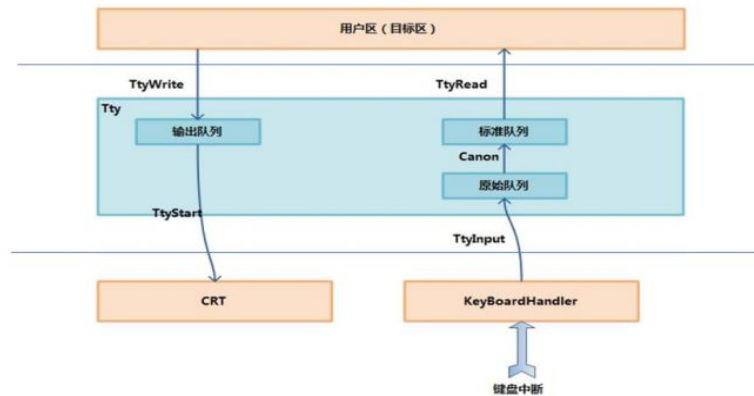
## 2 实验设备及工具

- 已安装 Windows 操作系统的 PC 机一台
- Bochs 虚拟机一台
- 集成开发环境：eclipse
- 编译器：MinGW
- 汇编器：NASM

## 3 基础知识

### 1: 字符终端的输入输出

- TTy 对象有 3 个队列，分别是标准队列 `t_canq`，原始队列 `t_rawq`，输出队列 `t_outq`。
- 队列的使用规则：原始队列 `t_rawq`，存放从键盘获得的原始输入字符。标准队列 `t_canq`，存放处理后的输入字符。标准的处理规则是：`backspace` 删除前面的字符，等等。输出队列 `t_outq`，存放向 CRT 输出的字符。
- TTy 是行缓存的，对应用程序而言，没有回车得不到输入，没有回车看不到输出。
- CRT 对象是一个屏幕控制对象，用于将 TTY 中输出队列的内容送入显存中，使之显示在终端屏幕上。
- `KeyboardHandler` 对象是一个键盘控制对象，用于将用户从键盘中输入的内容放入到 TTY 的原始队列中。

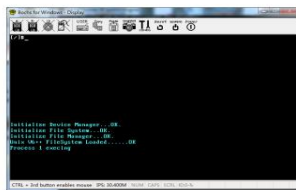


## 2: 界面显示

- 界面的显示就是 CRT 对象来做的. 它有两个核心变量, 一个是显存, 另一个是光标。  
显存可以显示 25\*80 个字符, 光标显示的是下一个显示字符的坐标。

### CRT类的核心变量

```
unsigned short* CRT::m_VideoMemory = (unsigned short *)
(0xB8000 + 0xC0000000);
```



```
unsigned int CRT::m_CursorX = 0;
unsigned int CRT::m_CursorY = 0;
char* CRT::m_Position = 0;
char* CRT::m_BeginChar = 0;
```

2018/4/20

UNIX系统分析 同济大学计算机系邓蓉

12

## 3: 键盘硬件和中断处理程序

- 一次敲击键盘, 产生 2 次中断。按下产生 1 个键盘中断, 放开产生第 2 个键盘中断。同一个按键, 2 次中断处理程序得到的数据是不一样的。
- 2、键盘控制器产生键盘中断的同时, 送数据入数据寄存器, 修改状态寄存器标记数, 据寄存器中有值需要读出
- 0x60 数据寄存器
- 0x64 状态寄存器
- 3、键盘中断处理程序, 识别用户输入信息: 字符, 送中断的输入缓存, 控制信号, 修改键盘工作模式, 或实施内核进程控制逻辑

## 4: Unix V6++的初始化过程

- 将磁盘上的引导扇区装入内存。
- 执行引导扇区中的代码：读磁盘，将操作系统内核模块装入内存。
- 初始化操作系统内核，主要工作包括：
  - 初始化中断向量表等与硬件相关的数据结构；
  - 初始化操作系统自身工作所需的全局变量；
- 创建以 0#、1#进程为代表的所有精灵进程（daemon）。

## 4 实验内容

### 1: 明确自己的目标

自己最终要实现的是两个终端，并且实现用户账户密码的登录，并且 Ctrl + t 切换终端。

### 2: 查看 Unix V6++的源码，理解单用户系统下用户是如何打开并绑定 tty 终端的。

- 绑定 tty 设备

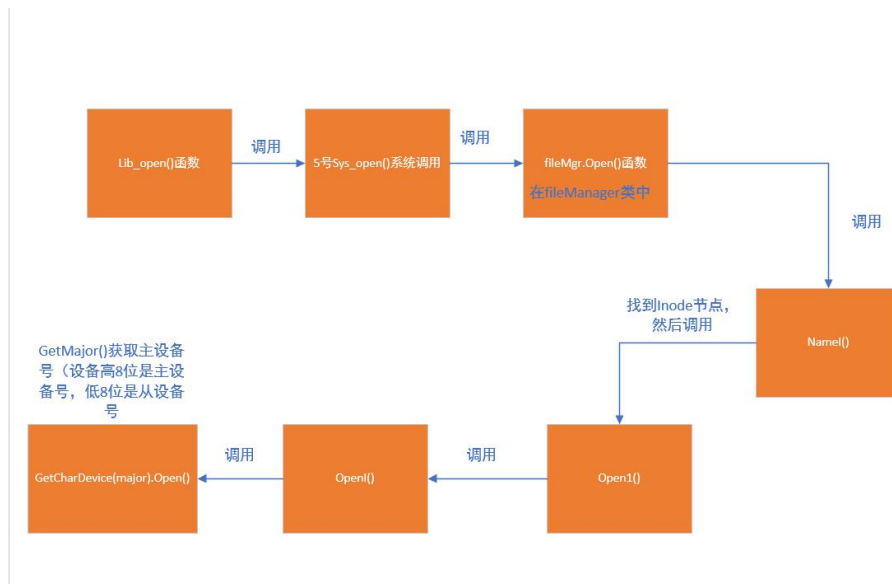
源代码 src/kernel/main.cpp 中，系统开机启动 0 号进程初始化 rootDirInode 和用户，当前工作目录后，分别以读和写两种方式打开 /dev/tty 设备。

```
/* 打开TTY设备 */
int fd_tty = lib_open("/dev/tty1", File::FREAD)

if ( fd_tty != 0 )
{
    Utility::Panic("STDIN Error!");
}
fd_tty = lib_open("/dev/tty1", File::FWRITE);
if ( fd_tty != 1 )
{
    Utility::Panic("STDOUT Error!");
}
```

这个是 STDIN 和 STDOUT 标准输入输出，之后分裂出的子进程会继承这两个文件描述符，可以通过这两个文件描述符进行输入输出。

- 通过阅读源码，可以得到系统打开 tty 终端的流程：



其中打开 `tty` 文件有两个文件描述符，

0 号文件描述符（STDIN），读打开方式打开 `tty` 文件。

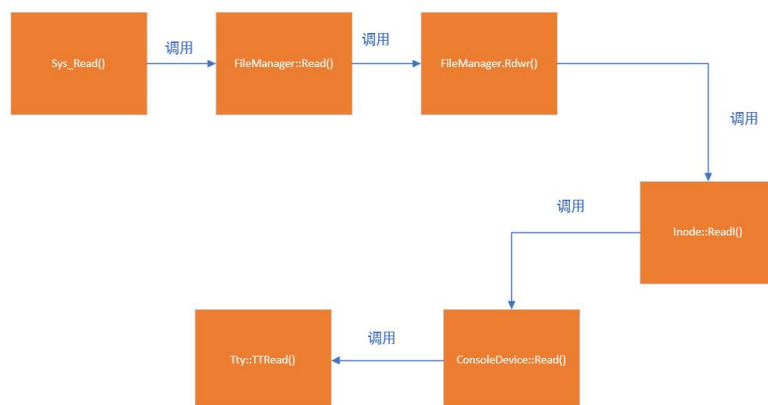
1 号文件描述符（STDOUT），写打开方式打开 `tty` 文件。

此后进程从 `STDIN` 中可以读取 `tty` 终端的输入，也可以通过 `STDOUT` 向 `tty` 终端中输出数据。

库函数 `printf()`, `gets()` 定义在头文件 `stdio.h` 中，是通过 `STDOUT`, `STDIN` 实现的。

### 3: TTy 终端的输入数据流程

举例：库函数 `gets()` 函数读取终端字符的流程：



此后，`TTY` 终端的输入字符就被读入到用户区供用户使用了。

`TTY` 终端输出数据流程与输入流程差不多，就是 `read` 函数变成 `write` 函数即可。

## 4: 终端显示 (CRT), 查看终端数据是怎么显示在屏幕上的。

1: 核心变量: 显存和光标。

在 CRT.cpp 中, 首先初始化核心变量。

```
unsigned short* CRT::m_VideoMemory = (unsigned short *) (0xB8000 + 0xC0000000);  
unsigned int CRT::m_CursorX = 0;  
unsigned int CRT::m_CursorY = 0;
```

2: CRTStart() 函数

根据判断字符是否为特殊标识来做出不通的举措。

```
while ( (ch = pTTY->t_outq.GetChar()) != TTY::GET_ERROR )  
{  
    switch (ch)  
    {  
        case '\n':  
            NextLine();  
            CRT::m_BeginChar = pTTY->t_outq.CurrentChar();  
            m_Position = CRT::m_BeginChar;  
            break;  
  
        case 0x15:  
            //del_line();  
            break;  
  
        case '\b':  
            if ( m_Position != CRT::m_BeginChar )  
            {  
                BackSpace();  
                m_Position--;  
            }  
            break;  
  
        case '\t':  
            Tab();  
            m_Position++;  
            break;  
  
        default: /* 在屏幕上回显普通字符 */  
            WriteChar(ch);  
            m_Position++;  
            break;  
    }  
}
```

由代码可知, “\n” 换行符执行 nextline () 函数换行, ‘\b’ 就执行 Backspace () 函数, 进行删除。其中 0x0A00 是显示光标的颜色。更改它可以换字体的颜色。

```

void CRT::BackSpace()
{
    m_CursorX--;

    /* 移动光标, 如果要回到上一行的话 */
    if ( m_CursorX < 0 )
    {
        m_CursorX = CRT::COLUMNS - 1;
        m_CursorY--;
        if ( m_CursorY < 0 )
        {
            m_CursorY = 0;
        }
    }
    MoveCursor(m_CursorX, m_CursorY);

    /* 在光标所在位置填上空格 */
    m_VideoMemory[m_CursorY * COLUMNS + m_CursorX] = ' ' | 0x0A00;
}

```

当输入的是普通字符的话, 就执行 WriteChar() 函数, 进行回显。

```

void CRT::WriteChar(char ch)
{
    m_VideoMemory[m_CursorY * CRT::COLUMNS + m_CursorX] = (unsigned char) ch | CRT::COLOR;
    m_CursorX++;

    if ( m_CursorX >= CRT::COLUMNS )
    {
        NextLine();
    }
    MoveCursor(m_CursorX, m_CursorY);
}

```

当字符满了一行之后, 就换行。

其中 ch 字符是由 TTy.cpp 中的 TTyOutput() 函数将字符放入输出字符缓存队列中。

```

/* 将字符放入输出字符缓存队列中 */
if (ch)
{
    this->t_outq.PutChar(ch);
}

```

## 5: 改写代码, 实现两个终端

### 1: 修改 CRT 的代码:

前面提到了光标变量, 我们需要实例化两个变量, 分别表示一个终端:

```

unsigned int CRT::m_CursorX = 0;
unsigned int CRT::m_CursorY = 0;
unsigned int CRT::m1_CursorX = variable;
unsigned int CRT::m1_CursorY = 0;
unsigned int variable = TTy::size;

```

其中 variable 是我自己定义的变量, 它最终是定义在 TTy.h 中的, 改变它可以



显示不同宽度的终端。它初始化第二个终端的 x 初始值。  
然后设置一个 ntty 变量，来具体表示是哪一个终端，根据终端来修改相应的坐标。例如：

```
void CRT::NextLine(int ntty)
{if(ntty == 0)
{
    m_CursorX = 0;
    m_CursorY += 1;

    /* 超出最大行数 */
    if ( m_CursorY >= CRT::ROWS )
    {
        m_CursorY = 0;
        ClearScreen(ntty);
    }
    MoveCursor(m_CursorX, m_CursorY);
}
else
{
    m1_CursorX = variable;
    m1_CursorY += 1;

    /* 超出最大行数 */
    if ( m1_CursorY >= CRT::ROWS )
    {
        m1_CursorY = 0;
        ClearScreen(ntty);
    }
    MoveCursor(m1_CursorX, m1_CursorY);
}
}
```

## 2: 修改 TTy 的代码

首先修改头文件: TTy.h, 我设置了一个当前终端号和一个 TTy 变量的当前终端号。

```
static const char GET_ERROR = -1,
static unsigned int TTy_no ; /*当前终端号*/

short dev; /* 土设备号 */
int ntty; /*当前终端号*/
// ...
```

## 3: 修改 CharDevice 的代码:

在头文件修改 TTy, 使它包含两个 TTy.

```
public:
    TTy* m_TTy[2]; /* 指向字符设备TTY结构的指针, 包含两个TTY */
};
```

在.cpp 中初始化 TTy 对象

```
/* 初始化TTY */
if ( NULL == this->m_TTy[minor] )
{
    this->m_TTy[minor] = &g_TTy[minor];
    m_TTy[minor]->ntty = minor;
}
```

Minor 是从设备号, 也就是当前对象的终端号。其他的做出相应的修改即可。

## 4 修改 Keyboard 代码

使它绑定当前的终端

```
TTY* pTTY = Kernel::Instance().GetDeviceManager().GetCharDevice(DeviceManager::TTYDEV).m_TTy[TTY::TTY_no];  
  
if ( NULL != pTTY)  
{  
    pTTY->TTYInput(ch);  
}
```

实现 Ctrl t 切换终端

```
}  
else if('t' == ch)  
{  
    TTY::TTY_no = TTY::TTY_no^1;  
    ch = 0;  
    CRT::FreshCurse(TTY::TTY_no);  
}  
else
```

就是把当前终端号切换，再执行 FreshCurse() 函数，改变坐标即可。

## 5 修改 shell.c

- 当 0 号进程打开/dev/tty1 设备后，fork 出 1 号进程，1 号进程继承了 0 号进程的文件描述符 STDIN 和 STDOUT，且 p\_ttyp 与 tty1 设备绑定。之后，0 号进程入睡，1 号进程执行 shell，然后 fork 出 2 号进程，2 号进程继承 1 号进程的文件描述符 STDIN 和 STDOUT，且 p\_ttyp 与 tty1 设备绑定。1 号进程继续 fork 出 3 号进程然后入睡，3 号进程先关闭继承过来的 0 号文件描述符和 1 号文件描述符，再以读方式打开/dev/tty2，这样 3 号进程的 0 号文件（STDIN）就绑定到了 tty2 设备上，以写方式打开/dev/tty2，3 号进程的 1 号文件（STDOUT）就绑定到了 tty2 设备上。这样 2 号进程和 3 号进程绑定的 tty 设备就各不相同了。
- 然后与终端进行交互实现账号密码的登入，我设定了两个账号（root 和 yy），账号密码通过之后，设置 uid 和 gid，例如 root 用户的 uid 为 0，gid 为 0，而普通用户的 uid 和 gid 为大于 500 的整数。最后进入[/]#状态，可以进行命令操作。  
进行密码输入的时候，为了保证安全性，输入的内容回显成\*\*\*\*\*，可以通过增加一个系统调用 Sys\_ReadPW 来实现

```
/* 49 = ReadPW count = 3 */  
int SystemCall::Sys_ReadPw(){  
    User& u = Kernel::Instance().GetUser();  
    u.u_procp->p_ttyp->t_flags=TTY::RPW;//设置TTY终端为读password模式  
    FileManager& fileMgr = Kernel::Instance().GetFileManager();  
    fileMgr.Read();  
    u.u_procp->p_ttyp->t_flags=TTY::ECHO;  
    return 0;  
}
```

此系统调用将 tty 终端的 flag 变为 RPW 模式，即数据回显位\*\*\*\*，读取完之后，再将 flag 变成 ECHO 模式，数据回显。再 TTy 对象的 TTyInput() 函数中添加：

```
else if(this->t_flags & TTy::RPW)
{ //读密码的话，回显*
    if(ch!='\n')
    {
        ch='*';
        this->TTyOutput(ch);
    }
    this->TTStart();
}
```

这样通过前面对 gets() 函数的梳理与启发，当你想读密码的时候回显\*\*\*，我在 stdio.c 中写了一个新的 gets() 函数 fgets() 来读密码。

```
}
void fgets(char* s)
{
    int n = rpw(STDIN, s, 1024);
    /* 以/n定界，因此最后要删去/n */
    if ( s[n - 1] == '\n' ) s[n - 1] = 0;
}
```

此后调用它读密码就显示为\*\*\*\*了。

```
if(t1==0 || t2== 0)
{ printf("password: ");
  fgets(password);
  printf("\n");
}
```

## 6 关于/dev/tty2 文件的创建问题。

/dev/tty2 文件是在生成虚拟机磁盘 c.img 时生成的：

```
cd ..\tools\MakeImage\bin\Debug && build.exe c.img boot.bin kernel.bin programs
copy ..\tools\MakeImage\bin\Debug\c.img "..\targets\UNIXV6++\c.img"
1 file(s) copied.
```

在 all 命令执行过程中，可以发现 build.exe 程序把 boot.bin（引导程序二进制文件）、kernel.bin（内核二进制文件）、program（应用程序）一起放入 c.img 文件中。而/dev/tty1 和/dev/tty2 文件正是在这个时候写入 c.img 中的。Build.exe 程序是一个 C#程序，在 Run.cs 文件中添加/dev/tty2 文件：

```
DevFile ttyDevFile = new DevFile(_superBlock, _inodeBlock, _dataBlock, _disk, "/dev/tty1", "tty");
ttyDevFile.CreateFile();

DevFile ttyDevFile1 = new DevFile(_superBlock, _inodeBlock, _dataBlock, _disk, "/dev/tty2", "tty2");
ttyDevFile1.CreateFile();

WriteWholeDir();
```

再在 DevFile.cs 文件中为/dev/tty2 程序添加设备号：

```

if(_devType == "tty")
{
    _fileInode._i_mode = (CommonFile.IALLOC | IFCHR | IREAD | IWRITE | IEXEC | (IREAD >> 3) | (IWRITE >> 3) | (IEXEC >> 3) | (IREAD >> 6)
    _fileInode._i_addr[0] = 0;
}
else if(_devType == "tty2")
{
    _fileInode._i_mode = (CommonFile.IALLOC | IFCHR | IREAD | IWRITE | IEXEC | (IREAD >> 3) | (IWRITE >> 3) | (IEXEC >> 3) | (IREAD >> 6)
    _fileInode._i_addr[0] = 1;
}

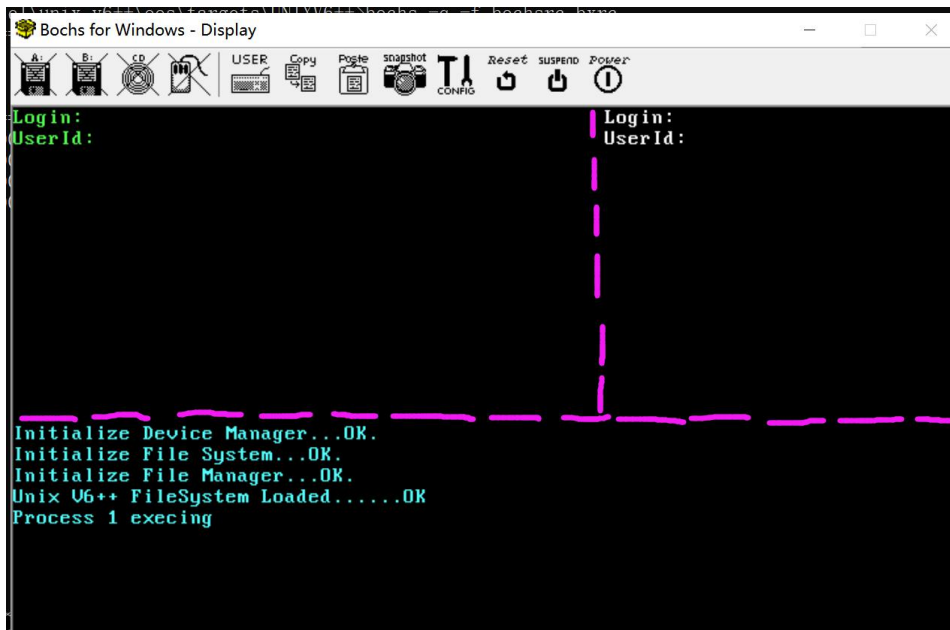
```

将修改好的 C#代码在 VS 下编译得到 exe 程序，将其移动到 oos/tools/MakeImage/bin/Debug 目录下即可。再次用 all 重新生成 c.img 即可。

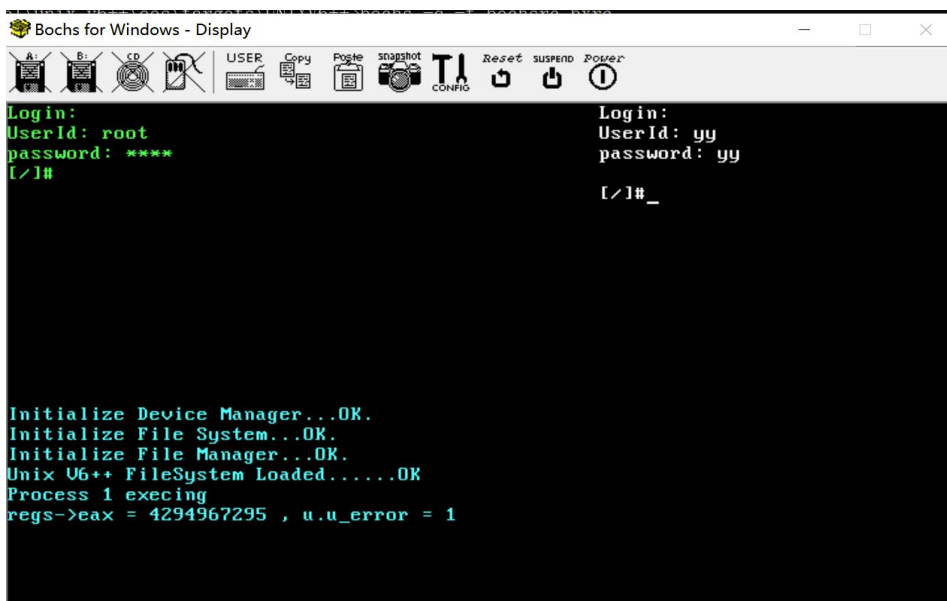
## 5 运行结果

- 具体实现了两个终端，每个终端的颜色不一样，所占宽度不一样。实现了账户密码登入以及密码隐藏（有一点 bug）。

1: 两个终端界面：



2: 用户登入：这里出现了一个错误，终端 2 的密码回显为\*\*失败。



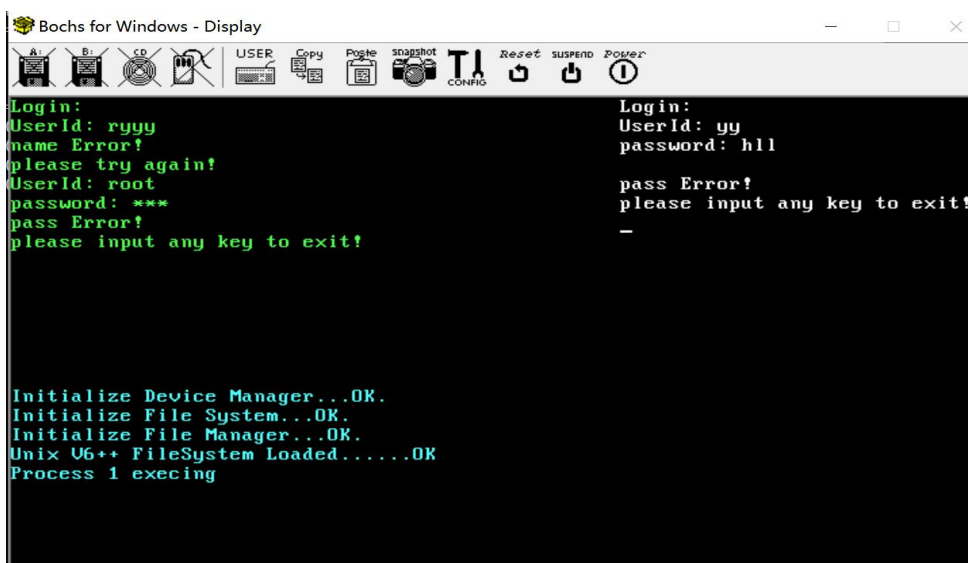
```
Bochs for Windows - Display
A: B: CD USER Copy Paste snapshot T1 Reset SUSPEND Power
CONFIG

Login:
UserId: root
password: ****
[/]#

Login:
UserId: yy
password: yy
[/]#_

Initialize Device Manager...OK.
Initialize File System...OK.
Initialize File Manager...OK.
Unix V6++ FileSystem Loaded.....OK
Process 1 execing
regs->eax = 4294967295 , u.u_error = 1
```

- 检测账号密码的正确性，当账号错误的时候可以重新再输入账号，密码错误之后直接按任意键结束。



```
Bochs for Windows - Display
A: B: CD USER Copy Paste snapshot T1 Reset SUSPEND Power
CONFIG

Login:
UserId: ryyy
name Error!
please try again!
UserId: root
password: ***
pass Error!
please input any key to exit!

Login:
UserId: yy
password: hll
pass Error!
please input any key to exit!
-

Initialize Device Manager...OK.
Initialize File System...OK.
Initialize File Manager...OK.
Unix V6++ FileSystem Loaded.....OK
Process 1 execing
```

## 6 总结反思

通过这次课设，了解了 Unix V6++ 的 TTY 终端输入输出原理，以及 CRT 终端显示，键盘输入，开机初始化的过程，其中最映像深刻的就是实现 shell 用户登入的交互，输入字符要与设定的字符对比用 `strcmp()` 函数去进行对比，然后对于密码的保护那一块，我 debug 了很久，就是终端 2 号哪里改它的 `TTY::t_flags` 为 `RPW` 的时候好像没有效果，所以终端 2 的密码没有实现隐藏。此外感慨一下，操作系统课设很好玩。