实验一 接管裸机的控制权 实验报告

数据科学与计算机学院　2017 级计算机科学与技术　王程钥

1　实验题目

接管裸机的控制权

2　实验目的

配置并应用实验环境

熟悉虚拟机的设置及使用方法

熟悉 NASM 编译器的语法

掌握创建空白软盘镜像的方法

掌握十六进制编辑器的使用方法

3　实验要求

3.1 搭建和应用实验环境

虚拟机安装，生成一个基本配置的虚拟机 XXXPC 和多个 1.44MB 容量的虚拟软盘，将其中一个虚拟软盘用 DOS 格式化为 DOS 引导盘，用 WinHex 工具将其中一个虚拟软盘的首扇区填满你的个人信息。

3.2 接管裸机的控制权

设计 IBM_PC 的一个引导扇区程序，程序功能是：用字符'A'从屏幕左边某行位置 45 度角下斜射出，保持一个可观察的适当速度直线运动，碰到屏幕的边后产生反射，改变方向运动，如此类推，不断运动；在此基础上，增加你的个性扩展，如同时控制两个运动的轨迹，或炫酷动态变色，个性画面，如此等等，自由不限。还要在屏幕某个区域特别的方式显示你的学号姓名等个人信息。将这个程序的机器码放进放进第三张虚拟软盘的首扇区，并用此软盘引导你的 XXXPC，直到成功。

4　实验方案

4.1 实验环境

编程环境：GCC+NASM

16 进制编辑器：WinHex，Hex Editor

虚拟机：WMware Workstation

虚拟机环境

- 操作系统　MS-DOS
- 内存　1MB
- 硬盘　102MB
- 处理器　1 核心
- 虚拟化引擎首选模式　Intel VT-x/EPT 或 AMD-V/RVI

4.2 NASM 汇编器的使用

NASM 的命令行格式如下

```
nasm -f bin myfile.asm -o myfile.com
```

以上命令会将 myfile.asm 编译成 myfile.com。

若编译错误，会返回 Error 或 Warning。若成功编译，则不返回信息。

4.3 创建空白软盘镜像

.flp 格式在 VMware 中被作为软盘镜像的一种，它的文件的内容为可执行程序在二进制下的编码，没有其他的额外编码。所以创建空白软盘的方式非常简单，建立相应大小的全 0 二进制文件即可。

以下为使用 C++创建空白软盘镜像的代码。

```
#include<bits/stdc++.h>
#define N 1440*1024
using namespace std;
char s[N*2];
int main()
{
  freopen("test.flp","w",stdout);
  memset(s,0,sizeof(a));
  fwrite(s,1,N,stdout);
}
```

4.4 使用十六进制编辑器修改镜像文件

在十六进制编辑器下打开编译好的 com 文件和空白 flp 镜像文件。将 com 文件全文复制到 flp 镜像文件的 0x0000 字节处，保证文件大小不变即可。

事实上，使用十六进制编辑器合并空白镜像与 com 文件太过麻烦。Flp 文件本质是一个二进制文件，修改工作本质上是在一段全 0 二进制文件的开头贴上 com 文件中的二进制编码，所以我编写了一段代码来完成这个过程。

以下为修改镜像文件的代码

```
#include<bits/stdc++.h>
#define N 1440*1024
using namespace std;
char s[N*2],ch;int tot=0;
int main()
{
  FILE *f1,*f2;
  f1=fopen("ceshi.com","rb");
  f2=fopen("ceshi.flp","wb");
  while(fread(&ch,sizeof(char),1,f1))s[tot++]=ch;
  fwrite(s,sizeof(char),N,f2);
  fclose(f1);fclose(f2);
  return 0;
}
```

4.5 使用虚拟机

创建一个空白虚拟机（具体配置已在 4.1 提及），并将修改后的 flp 文件加入到虚拟机的软盘镜像中，运行虚拟机即可。

5 实验过程

5.1 汇编程序的修改

我直接用 NASM 编译了老师给的 stone.asm 文件，随后编译器出现报错

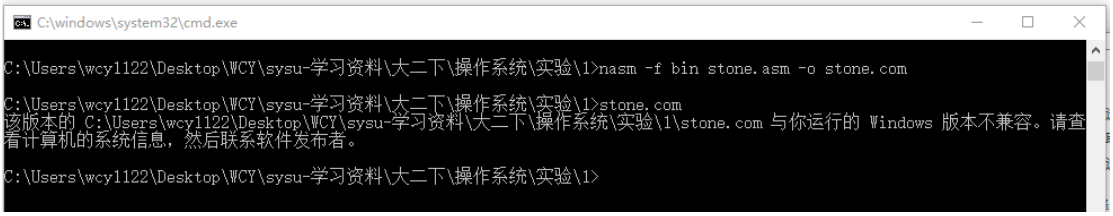
相关代码如下

```
.386

ASSUME cs:code,ds:code

code SEGMENT

code ENDS

END start
```

通过查阅资料，我发现这些代码都是 MASM 的格式。于是我将这些代码注释掉，并在数据定义部分前加上了 section .data，在正文前加上了 section .text，再编译即可编译成功。

5.2 本机运行 COM 文件

在编译成功后，我尝试直接打开.com 文件在 window10 环境下运行，结果却怎么都打不开。



64 位的 win10 并不兼容 16 位的 MS-DOS 应用程序，需要修改一些设置。

我尝试在网上搜索一些操作，比如在组策略中修改 16 位应用程序访问权限，比如打开注册表修改“DisallowedPolicyDefault”值。

但无论怎么修改都无济于事。于是我决定放弃本机运行，直接配置虚拟机。

5.3 Winhex 的使用

我将 com 文件复制到空白镜像文件后，却发现超过 200k 了无法保存。我查了一下，发现 Winhex 是一个付费软件，试用版只能用 45 天且无法保存超过 200k 的文件。

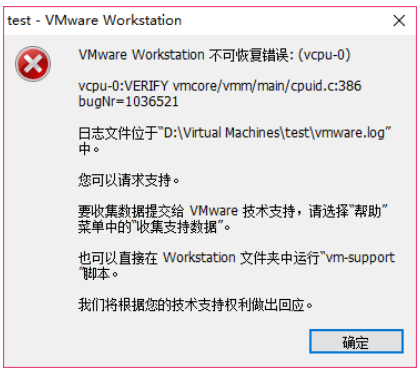
买不起正版 Winhex，也找不到好的破解方法。于是我决定找一个新的 16 进制编辑器。很幸运地，我找到了 Hex Editor 编辑器，并在该编辑器下完成了编辑。

后来我发现这样还是太麻烦，于是写了个简单的程序完成了编辑。代码见 4.4。

之后我在实验室的电脑上找到了破解版的 winhex14.2 软件，我直接把安装包拷贝到了自己的电脑，又重新安装了 winhex 软件。

5.4 虚拟机的配置

配置好虚拟机后，我准备开机运行，却发现 WMware 跳出了错误



我在同一个环境下测试了之前装过的 ubuntu 虚拟机，却发现可以运行。我去网上搜集了相关信息[4]，最后发现将处理器的虚拟化引擎的首选模式从自动改成 Intel VT-x/EPT 或 AMD-V/RVI 即可。

5.5 运行程序

我终于成功地进入了虚拟机。但令人遗憾的是，程序并没有如我所愿成功运行，屏幕上只显示了一个光标。

我回去查了一下汇编代码，发现 show 模块的代码有问题。

```
mov es:[bx],ax
```

这句话将输出的字符显示在 es+bx 的位置。但是在代码中 es 的初值是 0，并不是文本窗口显存起始地址 0B800h。代码中已经将 gx 赋值为对应的起始地址，所以我将 es 改成了 gs。

再装虚拟机，再运行，还是只显示一个光标。

我将输出'@'的代码修改成对应格式，将'@'定义到数据区，载入虚拟机后发现也无法显示。于是我注意到了一句话。

```
org 100h
```

我去网上查阅了一下 org 语句的意义，发现 org 语句起内存偏移的作用[2]。100h 是 com 文件的偏移位置，MS-DOS 裸机的内存偏移是 7c00h，于是我将 100h 改成了 7c00h，再载入虚拟机即可成功运行。

5.6 个性化拓展

5.6.1 字符改变

我想把’A’字符改成我的名字的简写字符串’wcy’。同时，我希望字符在每次碰壁都都进行修改，将’wcy’改成’ycw’，再改成’wcy’，依次类推。

为了实现这两个功能，我定义了两个字符串。

```
char1 db 'wcy'
char2 db 'ycw'
```

输出字符串是一个难题，我写了一个循环语句，使用 si 作为当前输出字符的循环变量，使用寄存器 cx 判断循环长度，成功输出了字符串。

```
mov cx, 3
print_str:
  mov byte al,[si]
  mov [gs:bx],ax
  inc si
  inc bx
  inc bx
  loop print_str
```

字符串修改的功能，我使用了寄存器 di，初值为 0。每次改变方向时 di 异或 1，即可实现 01 转换。在 di==0 时输出’wcy’，di==1 时输出’ycw’

以下为主要代码

```
mov si,char1
    inc di
    dec di
    jz print
    mov si,char2
```

5.6.2 颜色修改

我希望将输出的颜色改为彩色。我注意到每次输出前都有设置颜色的代码

```
mov byte ah,0Eh
```

这段代码将输出的颜色定为黑底白色，第一位为底，第二位为字体颜色。

我将其进行简单修改，在显示’wcy’时字体为黄色，在显示’ycw’时显示紫色。

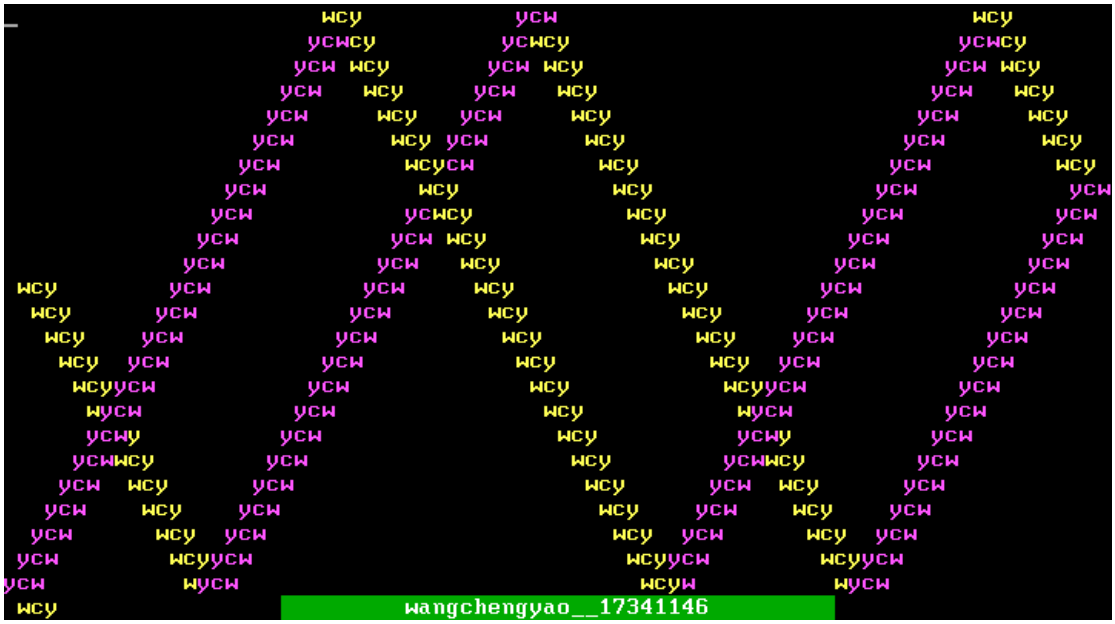
5.6.3 姓名学号输出

我将姓名学号输出在了最后一行，并将字符运动的范围进行了微小的修改。

关于字符串的输出在 5.6.1 已经提过，不再赘述。

5.6.4 反射问题

在改变了字符运动范围后，我发现了程序的一个 bug。当字符运动到边角后，可能会出界并停止运动。



如图所示，字符串进入到左下角碰壁后，向右翻转，随后出解（最后一方不是合法的运动区域）。

因此我重新阅读了一下代码，发现了一个问题。

```
DnRt:
    inc word[x]
    inc word[y]
    mov bx,word[x]
    mov ax,24
    sub ax,bx
      jz  dr2ur
```

```
    mov bx,word[y]
    mov ax,78
    sub ax,bx
      jz  dr2dl
    jmp show
dr2ur:
      mov word[x],23
      xor di,1
      mov byte[rdul],Up_Rt
      jmp show
dr2dl:
      xor di,1
      mov word[y],76
      mov byte[rdul],Dn_Lt
      jmp show
```

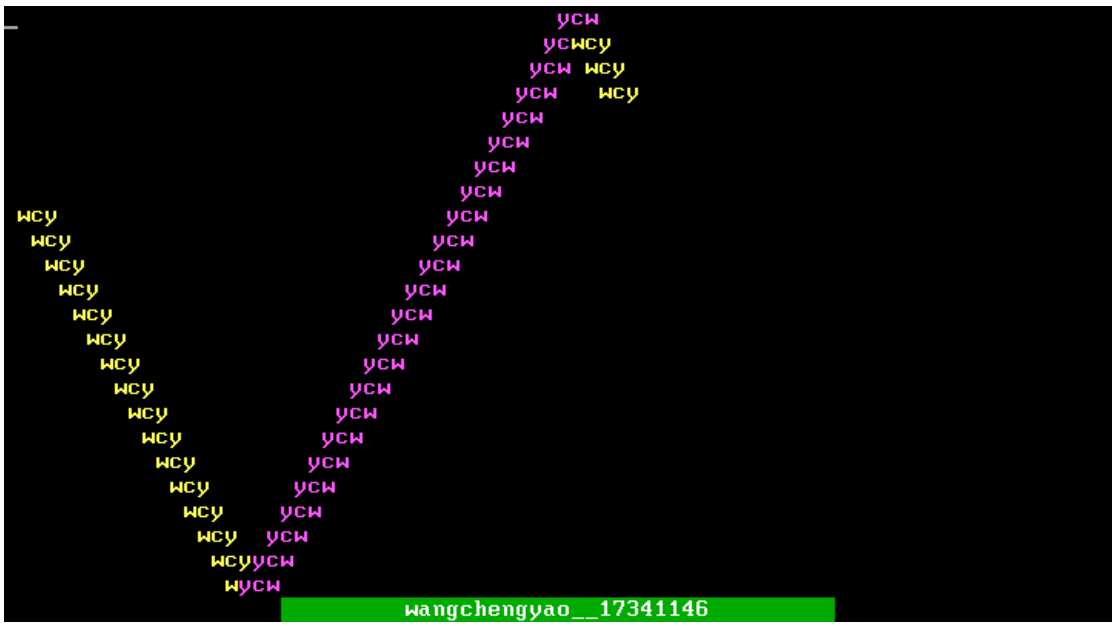
在每一次运动中，以右下方向（DnRt）为例，如果当前位置碰壁了，若是碰了下面则会向上反弹，若是碰了右面则会向左反弹。但若是碰到了右下角，它会向上反弹，但此时字符仍在界外。根据原程序，字符会在界外继续向右上方运动，这有违真实性。

所以我将代码中 dr2ur 模块进行了修改，若字符出了下界，若同时也出了右界，我会先让它向上反弹再让它向左反弹，后继续运动。

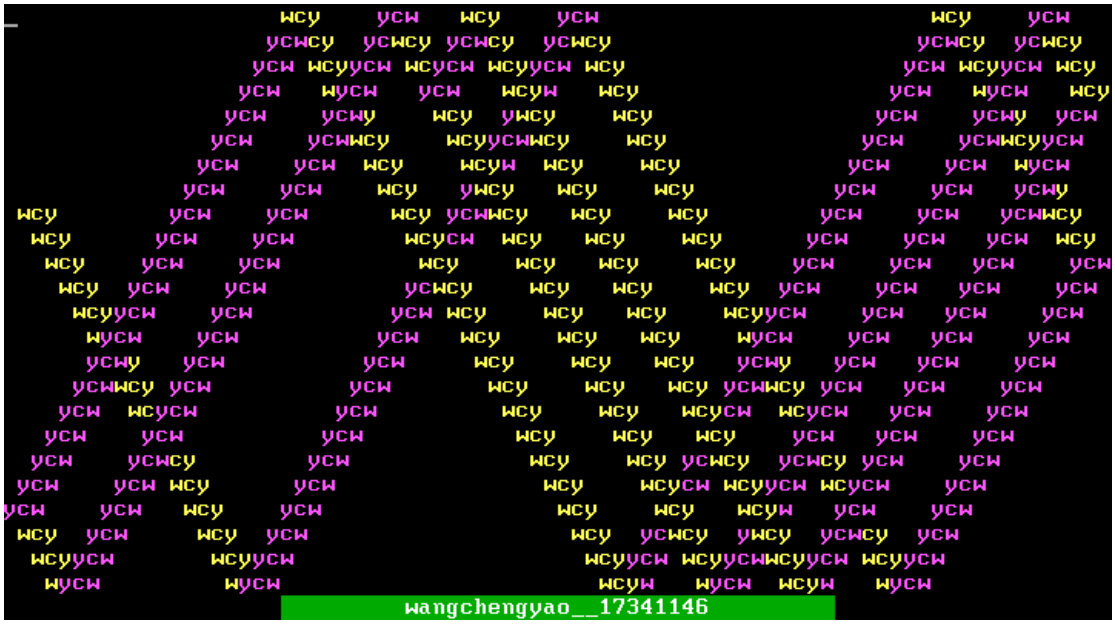
```
dr2ur:
      mov word[x],23
      mov bx,word[y]
      mov ax,78
      sub ax,bx
        jz  dr2dl
      xor di,1
      mov byte[rdul],Up_Rt
      jmp show
```

在修改之后，我发现.com 文件大小达到了 520KB，超过了 512KB，我删除了几条没用的指令，把它压到了 511KB，即可成功运行。

5.7 最终运行效果图

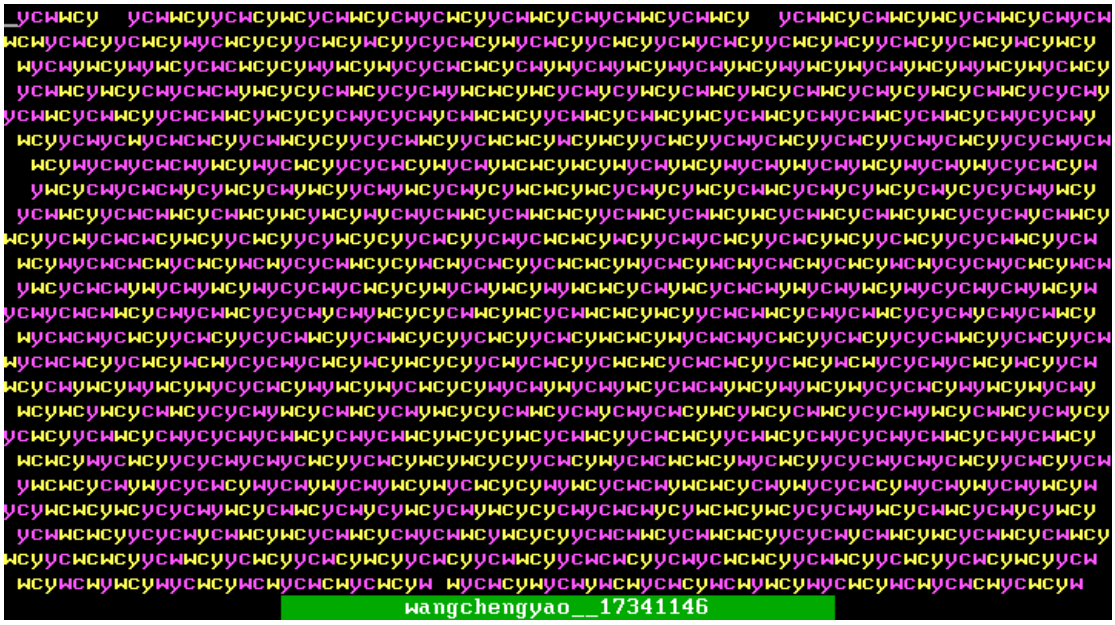


开始运行



运行一段时间

再运行一段时间



长时间运行后

6 实验总结

本次实验是操作系统实验这门课的第一次实验，要求搭建一套适合自己的实验环境，并应用这个环境接管裸机的控制权。之前的实验课老师都会给很多参考资料，甚至是每步的步骤。但这次实验是完全自己来，接触的也是一个之前完全没接触过的领域，从找环境到写代码都是全新的。因此这次实验可以说是摸着石头过河。

这次实验我遇到了很多问题，比如 NASM 和 MASM 的不同，比如看不懂汇编代码，比如运行不了虚拟机。在完全不熟悉的领域差错是一件很困难的事情，我也曾一度因为找不到

错，不会修改而崩溃，但还好最终没有放弃并完成了任务。我通过自己的尝试，自己查找资料，自己写代码测试，发现并解决了很多问题。上学期计算机组成原理学的是 MIPS 指令集，而本次实验一上来就是更复杂的 x86 指令集，我并没有系统学习过。为了看懂老师给的代码并进行修改，我在网上查找 x86 的资料，重新进行学习。

总之，本次实验我收获颇丰。通过这次实验，我了解了操作系统是怎么运行的，让我复习了汇编语言，熟悉了 16 进制编辑器和 VMware 的使用，也熟悉了操作系统相关的一些常识。更重要的是，这次实验锻炼了我的动手能力，解决问题能力，主动性，自主学习能力和探索能力。通过实践的亲身体会，我对课本知识也有了更深入地理解。

参考文献

[1]NASM 和 MASM
https://stackoverflow.com/questions/11572307/nasm-error-parsing-instruction-expected?lq=1 stackoverflow
[2]Org 语句 https://blog.csdn.net/mirage1993/article/details/29908929 csdn 博客
[3]汇编命令参考 https://blog.csdn.net/bjbz_cxy/article/details/79467688 csdn 博客
[4]VMware 不可恢复错误 https://blog.csdn.net/a6864657/article/details/82316417 csdn 博客