《操作系统原理》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | MTX | 学号 | U2021 | 专业班级 | 信安 | 时间 | 2023.11.21 |

1. **实验目的**

（1）理解操作系统引导程序/BIOS/MBR的概念和作用；

（2）理解并应用操作系统生成的概念和过程；

（3）理解并应用操作系统操作界面，系统调用概念

（4）掌握和推广国产操作系统（推荐银河麒麟或优麒麟，建议）

1. **实验内容**

（1）用NASM编写MBR引导程序，在BOCHS虚拟机中测试。

（2）在Linux（建议Ubuntu或银河麒麟）下裁剪和编译Linux内核，并启用新内核。（其他发行版本也可以）

（3）为Linux内核（建议Ubuntu或银河麒麟）增加2个系统调用，并启用新的内核，并编写应用程序测试。（其他发行版本也可以）

（4）在Linux （建议Ubuntu或银河麒麟或优麒麟） 或Windows下，编写脚本或批 处理。脚本参数1个：指定目录。脚本的作用是把指定目录中的全部文件的文件名 加后缀，后缀是执行脚本时的日期和时分。例如：文件名“test”变成“test2023-11-21-20-42”

1. **实验环境和核心代码**

**3.1 MBR引导程序编写测试**

首先，在Ubuntu虚拟机中安装BOCHS和nasm

先安装BOCHS环境：

sudo apt-get install build-essential xorg-dev libgtk2.0-dev

然后在NASM官网下载压缩包，解压后编译安装：

tar zxvf nasm-2.15.05.tar.gz

cd nasm-2.15.05.tar.gz

./configure

make

sudo make install

接下来安装Bochs，同样在官网下载，使用以下命令安装：

tar zxvf bochs-2.6.11.tar.gz

cd bochs-2.6.11.tar.gz

./configure --enable-debugger --enable-disasm

make

sudo make install

安装好后，接下来编写汇编代码，创建镜像，并将汇编代码编译如镜像中。

使用bximage生成镜像，如图1

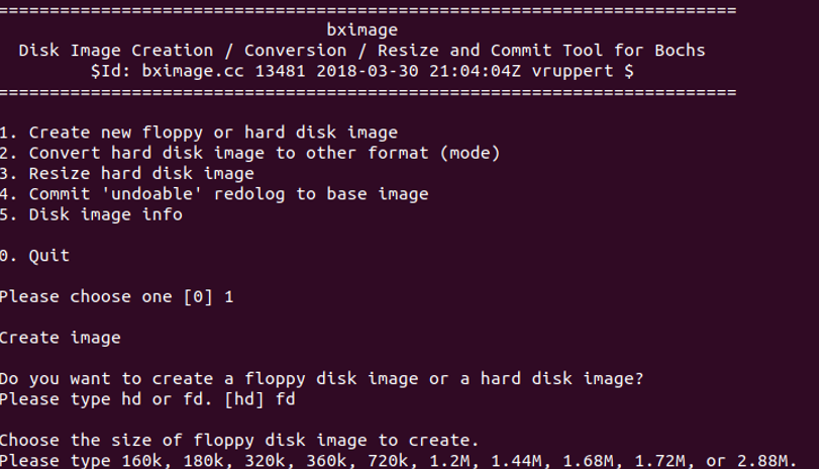


图1 bximage生成镜像

编写boot.asm

|  |
| --- |
| org 07c00h ; 告诉编译器程序加载到7c00处  mov ax, cs  mov ds, ax  mov es, ax  call DispStr ; 调用显示字符串例程  jmp $ ; 无限循环  DispStr:  mov ax, BootMessage  mov bp, ax ; ES:BP = 串地址  mov cx, 16 ; CX = 串长度  mov ax, 01301h ; AH = 13, AL = 01h  mov bx, 000ch ; 页号为0(BH = 0) 黑底红字(BL = 0Ch,高亮)  mov dl, 0  int 10h ; 10h 号中断  ret  BootMessage: db "Hello, OS world!"  times 510-($-$$) db 0 ; 填充剩下的空间，使生成的二进制代码恰好为512字节  dw 0xaa55 ; 结束标志 |

编译，把程序写入镜像：

|  |
| --- |
| nasm -f bin code.asm -o code.bin -l code.lst  dd if=code.bin of=a.img |

接下来要执行NASM代码，在文件夹里创建bochsrc.txt

|  |
| --- |
| megs:128  #模拟器的内存  romimage:file=/usr/local/share/bochs/BIOS-bochs-latest  #这个是BIOS-bochs-latest的路径，可能不一样  vgaromimage:file=/usr/local/share/bochs/VGABIOS-lgpl-latest  #这个是VGABIOS-lgpl-latest的路径，也可能不一样  floppya:1\_44=a.img,status=inserted  #这个是启动软盘,在当前目录下，如果不在当前目录，需要指明路径  boot:floppy  #表示从软盘启动  log:bochsout.txt  #日志输出文件 |

终端执行如下命令

bochs -f bochsrc

bochs

选择6开始模拟，如图2

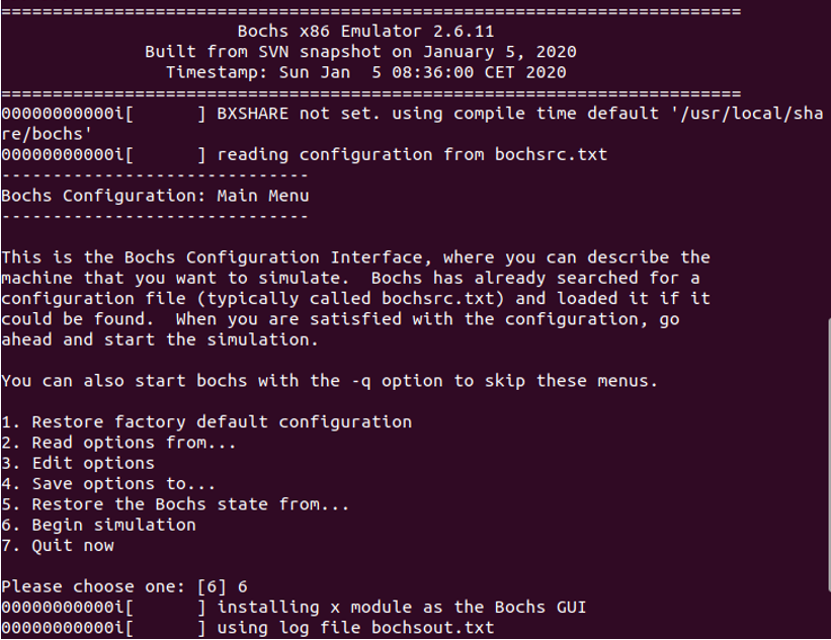


图2 开始模拟

这时出现黑窗口，再在终端输入字符‘c’，如图3

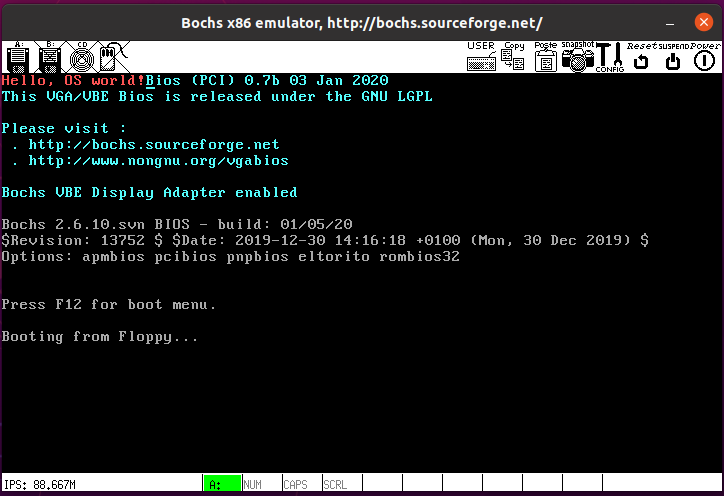


图3 成功打印字符串

可以看到，系统启动过程中执行了MBR区的代码，打印了字符串。

**3.2 & 3.3 ubuntu内核编译以及添加系统调用**

由于编译内核非常花时间，所以将任务2、3一起完成，实验效果是相同的，添加完系统调用后再编译。

先在Linux官网（https://www.kernel.org/）下载新的内核，本次实验使用的版本为5.15.4，下载后在用户文件夹下解压。

|  |
| --- |
| tar -xvf linux-5.15.4.tar.xz |

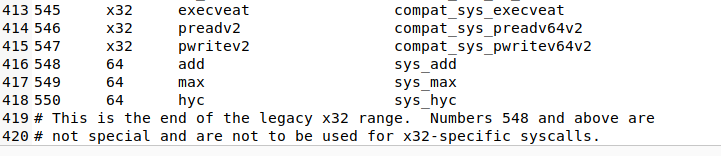
解压后使用如下命令安装依赖环境

|  |
| --- |
| sudo apt-get install libncurses5-dev openssl libssl-dev  sudo apt-get install build-essential openssl  sudo apt-get install pkg-config  sudo apt-get install libc6-dev  sudo apt-get install bison  sudo apt-get install flex  sudo apt-get install libelf-dev  sudo apt-get install zlib1g-dev |

接下来添加系统调用，先在系统调用表中添加调用号和函数名，进入内核文件夹，使用如下指令打开调用表：

|  |
| --- |
| sudo gedit arch/x86/entry/syscalls/syscall\_64.tbl |

在末尾按照序号添加系统调用add，max：

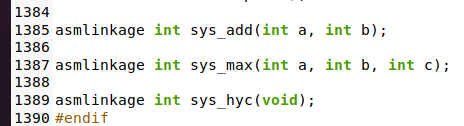


注意，不同版本的内核系统调用的数量不一样，按照上面的格式写就行。

使用如下命令，打开系统调用函数头文件

|  |
| --- |
| sudo gedit include/linux/syscalls.h |

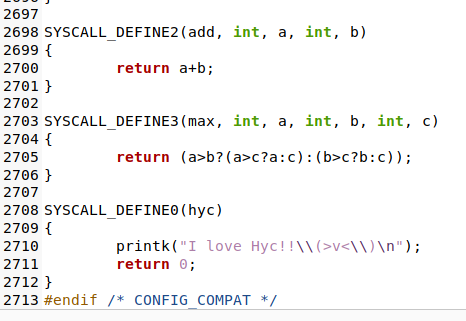
添加函数声明：



编写函数原型，在如下文件中编写：

|  |
| --- |
| sudo gedit kernel/sys.c |

同样在末尾添加：



注意，这里SYSCALL\_DEFINE后面的数字是函数参数个数，一个参数就是1，没有参数就是0。后面括号里跟的，第一个是函数名，后面的是参数类型和参数名。最后一个调用中，打印使用的是printk，因为没有c基本输入输出头文件，功能是向内核信息区写入信息，可以在终端中使用sudo mesg命令查看。

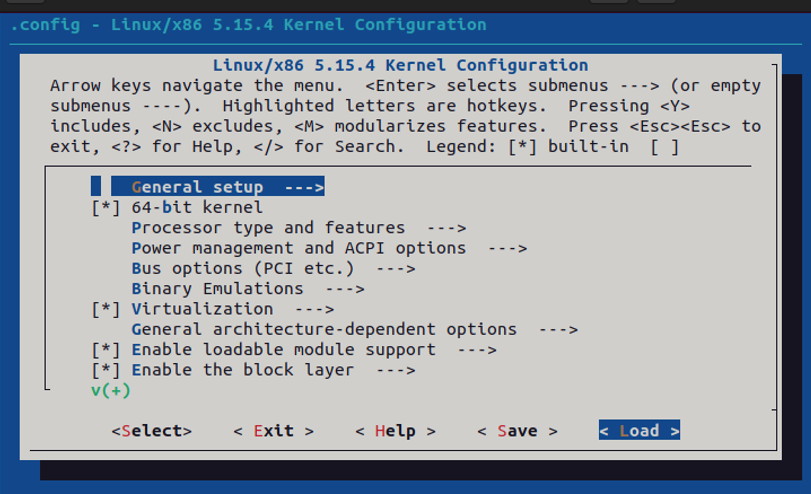
接下来开始编译，先创建配置文件，可以直接使用当前使用内核的配置文件，在boot文件夹下找到版本最高的内核版本的配置，复制到内核文件夹下的.config文件。

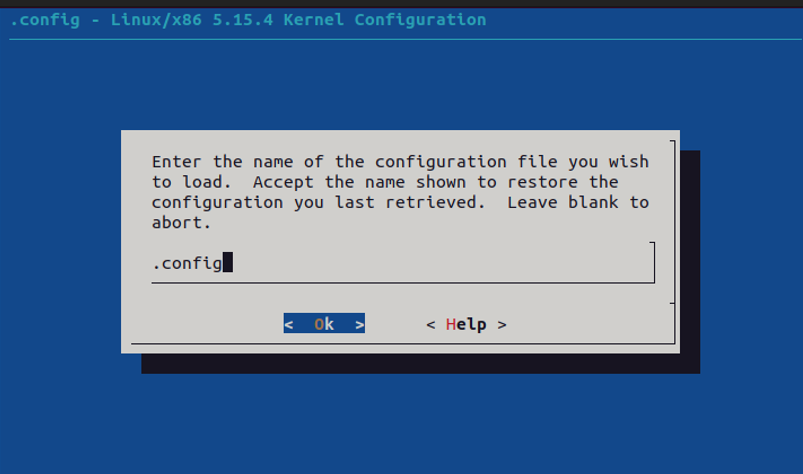
|  |
| --- |
| sudo cp /boot/config-5.11.0-40-generic ./config |

然后控制台输入

|  |
| --- |
| sudo make menuconfig |

跳出图形化配置界面，选择load，输入.config，选择OK，最后save





配置好后开始编译

|  |
| --- |
| sudo make -j16  sudo make modules\_install  sudo make install |

这里第一条命令-j后面的数字是处理器逻辑核数，虚拟机分配了几个核就用这个数的两倍。配置文件错了就改一下然后重新编译，重新编译前记得清除之前的文件，使用如下命令：

|  |
| --- |
| sudo make mrproper  sudo make clean |

全部编译成功后，重启。

进入Ubuntu高级选项界面，即可见到编译好的内核，选择该内核启动。

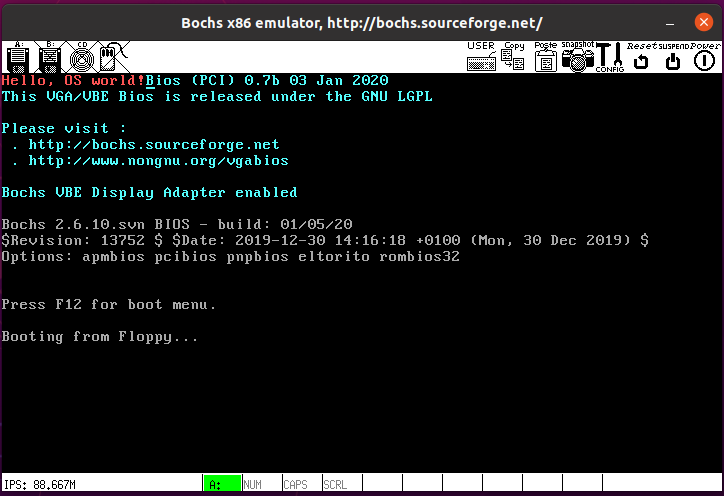
* 1. **批处理文件**

编写脚本文件如下：

|  |
| --- |
| #!/bin/bash  # 指定目录路径  directory=""D:\\桌面""  # 获取当前日期和时分  suffix=$(date +"%Y-%m-%d-%H-%M")  # 遍历目录中的所有txt文件  for file in "$directory"/\*.txt; do  # 提取文件名和扩展名  filename=$(basename "$file")  extension="${filename##\*.}"  filename="${filename%.\*}"  # 添加日期和时分后缀  new\_filename="${filename}-${suffix}.${extension}"  # 检查新文件名是否已存在，如果存在则更新后缀  counter=1  while [[ -e "$directory/$new\_filename" ]]; do  new\_filename="${filename}-${suffix}-${counter}.${extension}"  counter=$((counter+1))  done  # 重命名文件  mv "$file" "$directory/$new\_filename"  done |

1. **实验结果**

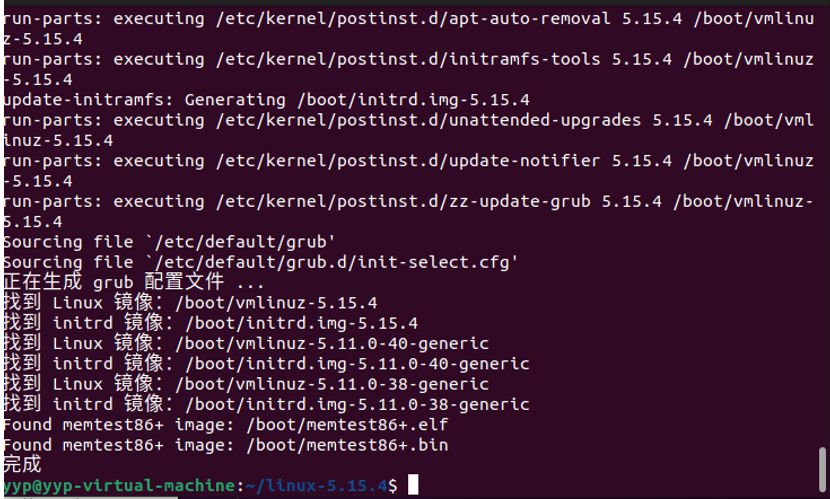
**4.1 MBR引导程序编写测试**



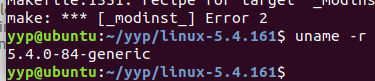
可以看到，系统启动过程中执行了MBR区的代码，打印了字符串。

**4.2 & 4.3 ubuntu内核编译以及添加系统调用**

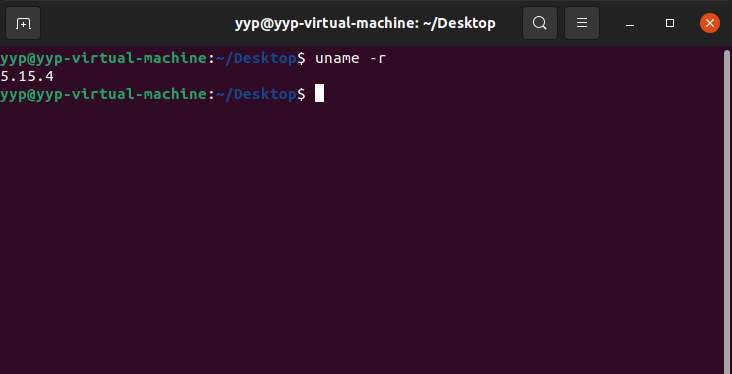
编译完成后显示如下：



当前版本：



重启选用新内核



可以看到，内核版本已升级至5.15.4。

对于系统调用，编写如下c程序进行验证

|  |
| --- |
| #include <stdio.h>  #include <linux/kernel.h>  #include <sys/syscall.h>  #include <unistd.h>  int main()  {  /\*548 549 550\*/  int a, b, c, d;  a=10;b=20;c=30;  d = syscall(548,a,b);  printf("%d\n",d);  d = syscall(549,a,b,c);  printf("%d\n",d);  d=syscall(550);  return 0;  } |

编译运行如下，符合预期。



**4.4批处理文件**



通过测试，可以看出运行脚本后，txt文件都加上或更新了时间后缀，符合预取。

1. **实验错误排查和解决方法**

**5.1 MBR引导程序编写测试**

设置BOCHS虚拟机的配置文件遇到了许多问题，我要确保正确指定了引导扇区文件的路径，并根据需要调整内存大小和其他配置选项，通过查看BOCHS文档或参考资料以获取更多关于配置文件选项的信息的方法解决了问题。

对BIOS终端不熟悉，要求使用BIOS中断来实现特定功能，如屏幕显示和获取内存信息，通过参考相关的BIOS中断文档或参考资料来了解不同中断的功能和使用方法解决了问题

。

**5.2 & 5.3 ubuntu内核编译以及添加系统调用**

在进行内核编译过程中，遇到许多错误。编译过程中的错误涉及缺少必要的文件或配置文件，导致编译无法完成。根据错误信息在网上查找相关信息，阅读官方文档即可解决。

**5.4批处理文件**

bash脚本比较简单，语法网上一搜就有，所以没什么问题。

1. **实验参考资料和网址**

<https://www.cnblogs.com/chengmf/p/12526821.html>

https://www.cnblogs.com/LyShark/p/13353400.html

https://blog.csdn.net/weixin\_44224230/article/details/89945899