《操作系统原理》实验报告

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 |  | 学号 |  | 专业班级 |  | 时间 | 2023. |

**一、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。

**二、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下裁剪和编译Linux内核，并启用新内核。
3. 为Linux内核增加2个系统调用，并启用新的内核，并编写应用程序测试。
4. 在Linux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“test-2023-11-21-20-42”。

**三、实验环境和核心代码**

**3.1 MBR引导程序编写**

实验环境：

操作系统：docker ubuntu:18.04

主要工具版本：NASM 2.13.02，GNU Make 4.1，Bochs 2.6

MBR程序启动代码如下

1. section .text
2. global \_start
3. \_start:
4. org 0x7C00
5. mov ax, boot\_msg
6. mov bx, 0x0009
7. call print\_str
8. call print\_endl
9. mov ax, mem\_size\_msg
10. mov bx, 0x0002
11. call print\_str
12. call get\_mem\_size
13. call print\_hex
14. boot\_end:
15. jmp boot\_end

虚拟机启动后，MBR程序被加载到0x7C00地址处，调用子函数获取内存大小信息，打印字符串信息，程序执行结束后进入循环。

获取内存大小信息代码如下

1. get\_mem\_size:
2. mov ah, 0x88    ; get extended memory (kb)
3. **int** 0x15
4. add ax, 0x400   ; 1M basic memory
5. ret

使用BIOS功能号0x15的子功能0x88获取扩展内存大小，即大于1MB的内存大小，以kb为单位。

数据段定义

1. boot\_msg:   db "Hello Noxke!", 0
2. mem\_size\_msg:   db "Memory Size:", 0
3. times   510-($-$$)  db 0
4. dw  0xAA55  ; boot singature

数据段定义了程序使用的字符串，并填充代码长度到510字节，最后定义SMB启动表示0xAA55，使虚拟机可以将编译后的镜像作为启动磁盘。

编写Bochs虚拟机配置文件

1. megs: 2
2. romimage: file=bochs/BIOS-bochs-latest
3. vgaromimage: file=bochs/VGABIOS-lgpl-latest
4. floppya: 1\_44="Image", status=inserted
5. boot: floppy
6. log: bochs\_log.txt
7. cpu: count=1, ips=4000000

**3.2 Linux内核的编译和启用**

实验环境：

操作系统：Ubuntu 20.04 LTSC linux-5.4.0

内核版本：linux-6.7.0-rc2

编译工具链：make libncurses-dev libssl-dev libelf-dev bison flex bc dwarves dracut kmod update-grub2

操作步骤：

make localmodconfig 生成.config编译配置文件

sudo make -j 8 开始编译内核

sudo make modules\_install 安装驱动模块

sudo make install 安装编译后的新内核

重启系统在GRUB菜单中选择新的内核

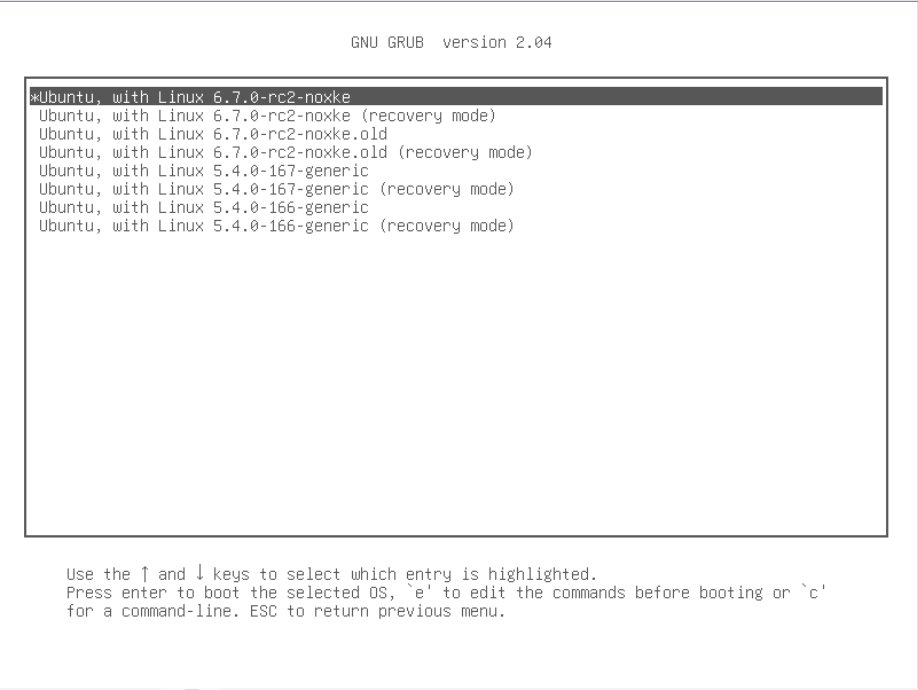


图3-1 GRUB菜单选择新的Linux内核

**3.3 增加Linux内核系统调用**

修改arch/x86/entry/syscalls/syscall\_64.tbl文件，添加系统调用号

1. # my\_syscall
2. 548     64      my\_add              sys\_my\_add
3. 549     64      my\_max              sys\_my\_max

添加头文件include/linux/noxke\_dbg.h

1. #ifndef NOXKE\_DBG\_H
2. #define NOXKE\_DBG\_H
4. # include <linux/syscalls.h>
6. // my\_syscalls
7. asmlinkage **long** sys\_my\_add(**int** a, **int** b);
8. asmlinkage **long** sys\_my\_max(**int** a, **int** b, **int** c);
10. #endif

头文件中声明系统调用函数sys\_my\_add和sys\_my\_max

添加代码模块kernel/noxke\_dbg.c

1. #include <linux/noxke\_dbg.h>
3. // my\_syscalls
4. SYSCALL\_DEFINE2(my\_add, **int**, a, **int**, b)
5. {
6. **return** a + b;
7. }
9. SYSCALL\_DEFINE3(my\_max, **int**, a, **int**, b, **int**, c)
10. {
11. **int** max = a;
12. **if** (b > max)
13. {
14. max = b;
15. }
16. **if** (c > max)
17. {
18. max = c;
19. }
20. **return** max;
21. }

在noxke\_bdg.c模块中实现上述声明的系统调用函数。

修改kernel/Makefile文件，将noxke\_dbg模块添加到内核中

1. obj-y += noxke\_dbg.o

重新编译安装内核

编写代码测试添加的系统调用

1. #include <stdio.h>
2. #include <sys/syscall.h>
4. #define \_\_x64\_sys\_my\_add 548
5. #define \_\_x64\_sys\_my\_max 549
7. **int** main()
8. {
9. **int** a = 123;
10. **int** b = 98;
11. **int** c = 230;
12. **int** add = syscall(\_\_x64\_sys\_my\_add, a, b);
13. **int** max = syscall(\_\_x64\_sys\_my\_max, a, b, c);
14. printf("%d + %d = %d\n", a, b, add);
15. printf("max(%d, %d, %d) = %d\n", a, b, c, max);
16. **return** 0;
17. }

**3.4 Linux shell脚本编写**

首先对命令行路径参数进行处理，检查路径是否存在，并处理路径末尾的‘/’

1. path=$1
3. # 检查目录是否存在
4. # path末尾删除/
5. **if** [ -d "$path" ]
6. then
7. **if** [[ $path =~ .\*/$ ]]
8. then
9. path=${path%/}
10. fi
11. **else**
12. echo "path error"
13. exit 1
14. fi

获取脚本运行时间戳，并处理为格式化字符串

1. datetime=`TZ="Asia/Shanghai"  date +%Y-%m-%d-%H-%M`
3. echo $datetime

遍历目录中的文件，使用正则表达式检测文件名是否含有时间戳，若不含时间戳则将文件名末尾添加时间戳，若存在时间戳则更新时间戳内容

1. # 遍历目录
2. **for** file in "$path"/\*;
3. **do**
4. # 只对文件进行操作
5. **if** [ -f $file ]
6. then
7. # ls -lh $file
8. # 识别文件末尾是否有时间后缀
9. **if** ! echo "$file" | grep -qE "^.\*(-[0-9]+){5}"
10. then
11. # 添加文件后缀
12. mv $file "$file-$datetime"
13. **else**
14. # 修改后缀
15. file\_name=`echo $file | sed -E 's/(-[0-9]+){5}$//'`
16. mv $file "$file\_name-$datetime"
17. fi
18. fi
19. done

**四、实验结果**

**4.1 MBR引导程序编写**

编译代码生成可启动的镜像文件

1. nasm -o boot.o boot.s
2. dd **if**=boot.o of=Image bs=512 conv=sync

启动Bochs虚拟机测试

bochs -q -f bochsrc.bxrc

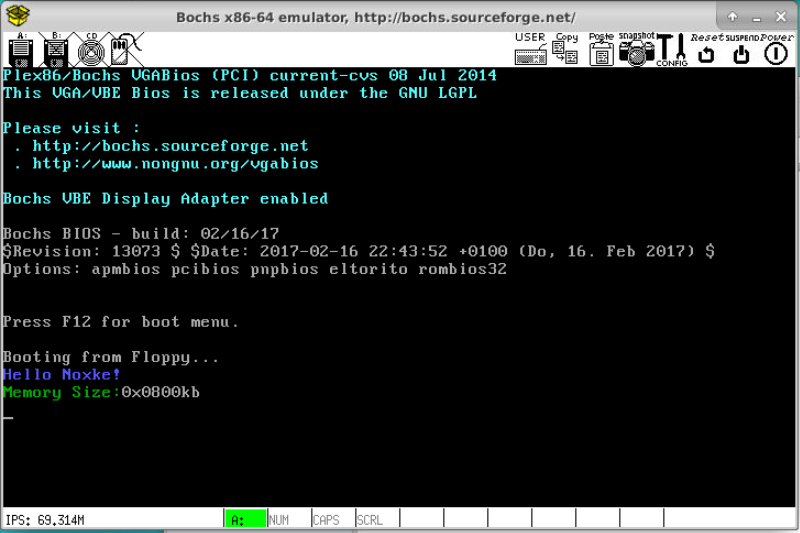


图4-1 MBR引导程序启动测试

如图4-1所示，虚拟机启动后运行了编写的MBR程序，打印提示字符串，打印系统内存大小0x0800kb(2MB)。

**4.2 Linux内核的编译和启用**

完成内核编译安装后，重启系统在GRUB菜单中选择新的内核

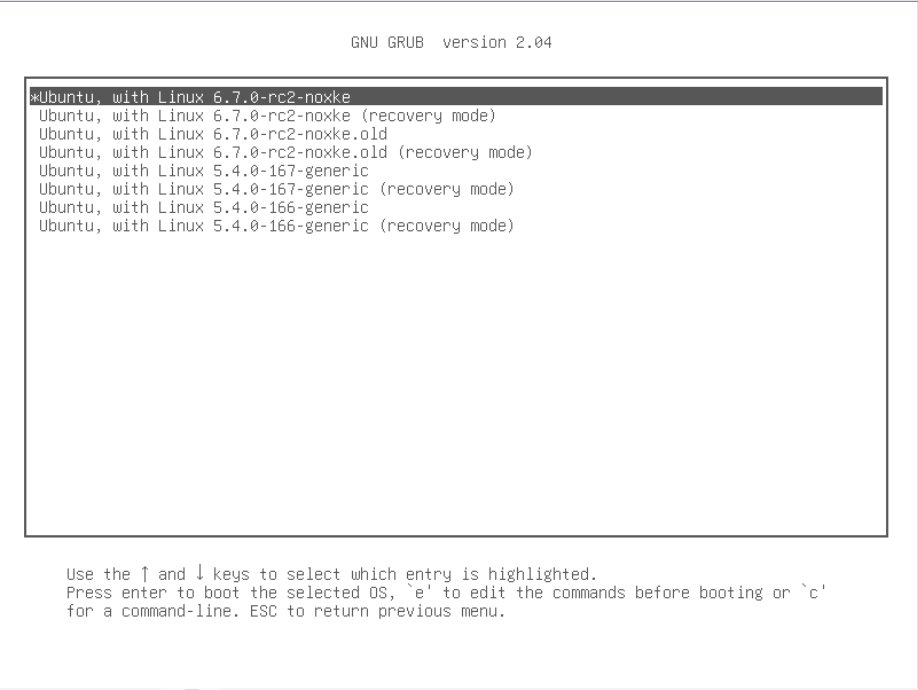


图4-2 GRUB菜单选择新的Linux内核

重启后使用uname目录查看内核版本，如图4-3所示，系统内核已更换为6.7.0-rc2

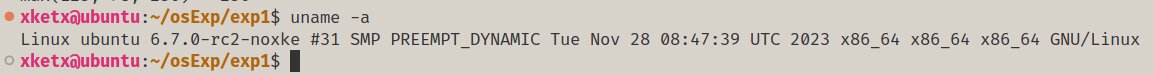


图4-3 uname查看内核版本

**4.3 增加Linux内核系统调用**

编译并执行系统调用测试程序，如图4-4所示，sys\_my\_add和sys\_my\_max可以正常调用并能够返回正确结果。

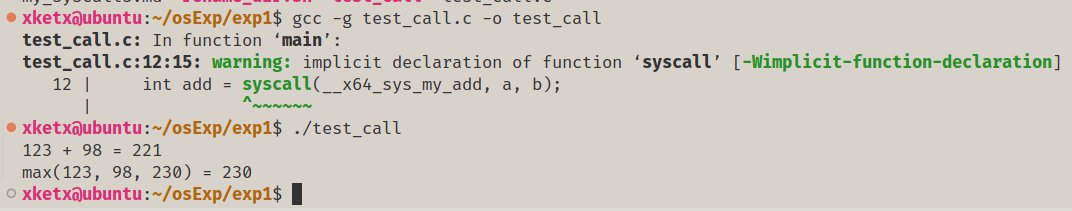


图4-4 系统调用测试

**4.4 Linux shell脚本编写**

创建测试文件夹，添加文件进行测试，如图4-5



图4-5 测试文件夹

运行shell脚本，如图4-6所示，无时间戳的文件末尾添加了时间戳，原有时间戳后缀的文件末尾时间戳被更新，文件夹名保持不变。

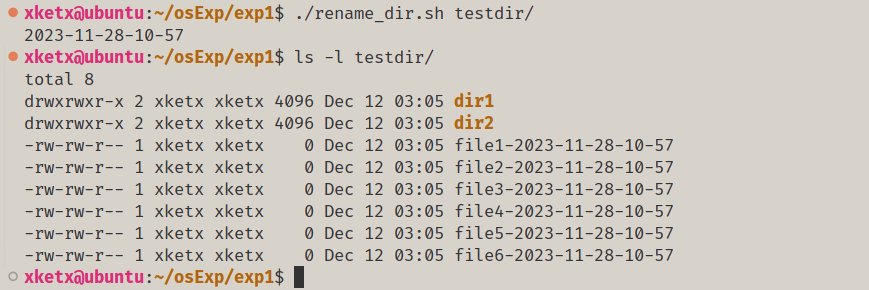


图4-6 shell脚本运行结果

**五、实验错误排查和解决方法**

**5.1 MBR引导程序编写**

使用0x15调用的0x88功能号获取系统内存时，发现获取到的内存与配置文件中分配的内存大小不符，多次修改配置文件发现获取到的值始终比实际内存小0x400kb，查询BIOS调用资料发现，该功能号获取到的是系统扩展内存，只能获取大于1M的部分，需要加上0x400kb才是系统总内存。

使用BIOS功能号输出字符串时，最开始选择使用0x10调用的0x0E功能号逐个字符输出，虽然能够实现字符串输出功能，但是使用该方法输出的字符无法定义颜色，如图5-1所示。

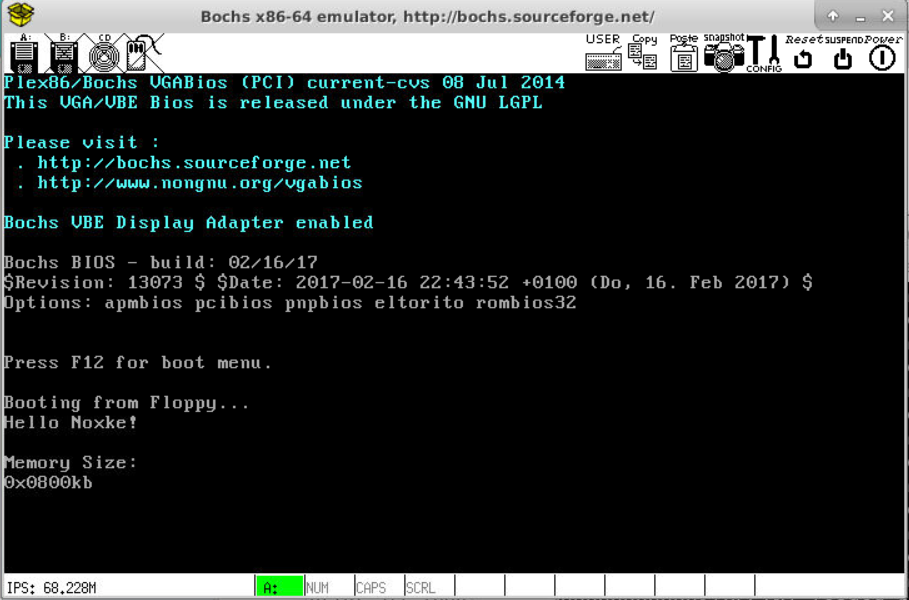


图5-1 逐字符输出字符串

将功能号修改为0x13直接输出字符串，可以定义字符串颜色，并且可以直接输出完整字符串，但是测试时发现只能输出最后一次的字符串，并且位置并不在最末行（如图5-2所示），单步调试测试发现第一次字符串能够输出，但是会被第二次输出的字符串覆盖。

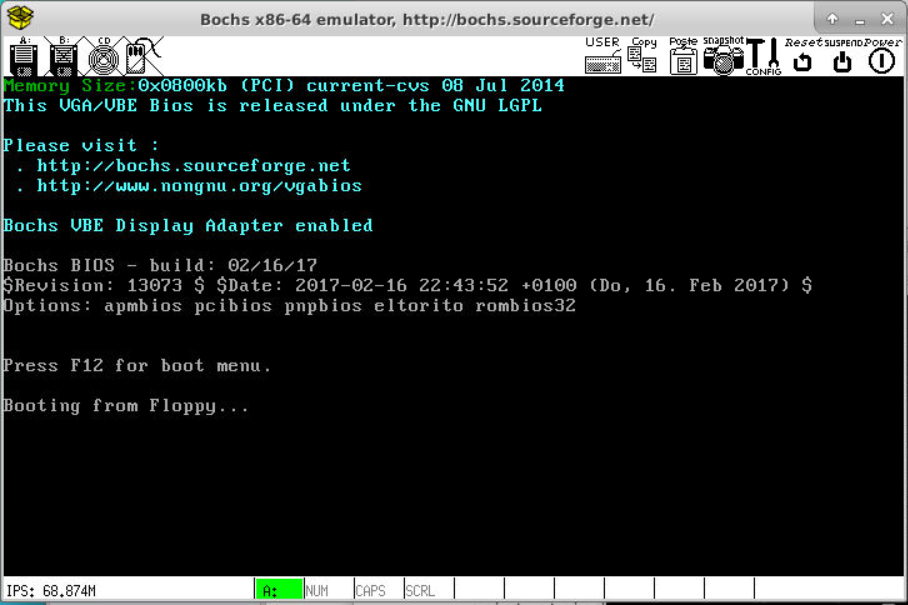


图5-2 0x13输出字符串

查询相关资料发现0x13功能号需要在dx寄存器中设置光标位置，输出字符串时会从光标指定的位置开始，可以手动设置光标位置，也可以使用0x10调用的0x03功能号自动获取光标位置，添加获取光标位置的调用后，MBR程序的字符串输出在了正确的位置（如图5-3所示）。

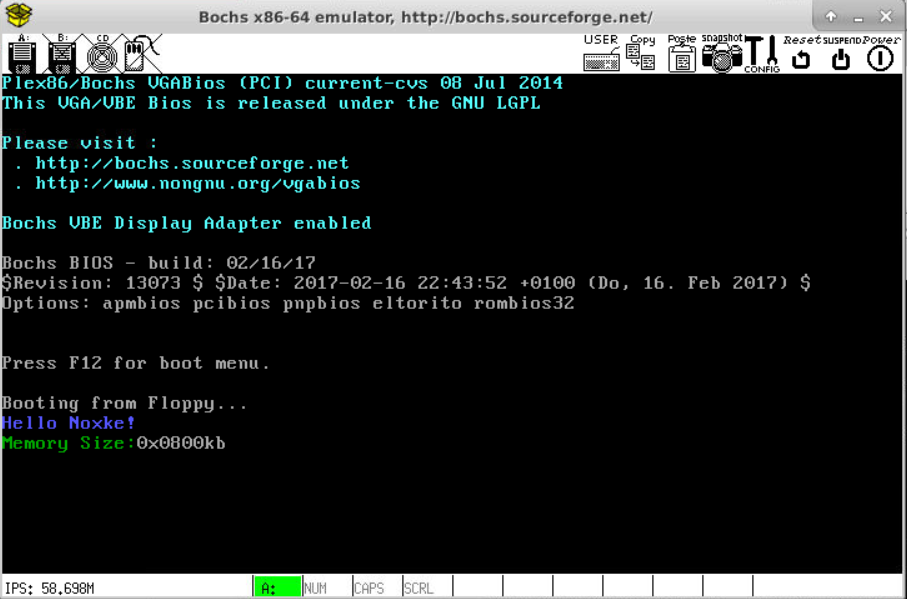


图5-3 0x03获取光标位置

**5.2 Linux内核的编译和启用**

在编译内核代码时，最开始准备使用docker容器配置编译环境，编译完成后将编译生成的内核镜像和驱动文件添加到Ubuntu系统中，更新grub后启动时发现始终无法启动，会在文件系统挂载时停止运行，反复测试后猜测是缺少驱动导致的，在Ubuntu中配置工具链重新编译。

在配置编译选项生成.config文件时，使用了menuconfig的默认配置进行编译，在使用8个线程编译的情况下，编译用时在3个小时左右，编译时间过长，内核能够正常启动，并且功能正常，但是发现编译生成的vmlinux镜像大小大于400M，驱动文件超过10gb，查看编译日志发现所有的驱动模块都被编译。

重新使用localmodconfig根据系统需要选择驱动模块生成.config配置文件，编译时长缩短到了10分钟左右，编译后生成的vmlinux镜像大小正常，驱动模块大小也恢复正常。

**5.3 增加Linux内核系统调用**

在noxke\_dbg.h头文件中声明了新的系统调用后，在noxke\_dbg.c中实现系统调用函数时，想当然地使用了声明时的格式asmlinkage long my\_add来定义函数，编译完成后的链接阶段，发生sys64\_my\_add无法找到的错误，反复修改头文件，系统调用声明都未能解决该问题，仔细翻阅官方文档发现系统调用函数需要使用SYSCALL\_DEFINEn宏进行定义，修改系统调用定义后，才成功完成链接操作，后续功能测试正常。

**5.4 Linux shell脚本编写**

使用grep正则表达式时未注意-e选项和-E选项的区别，使用-e选项匹配时无法匹配到正确结果，查询help文档发现-e选项正则表达式无法匹配通配符，需要使用-E选项进行高级正则表达式匹配。

**六、实验参考资料和网址**

**（1）教学课件**

**（2）Bochs虚拟机配置文件编写**

https://bochs.sourceforge.io/doc/docbook/user/bochsrc.html

1. **Linux新增系统调用**

https://www.kernel.org/doc/html/latest/process/adding-syscalls.html