

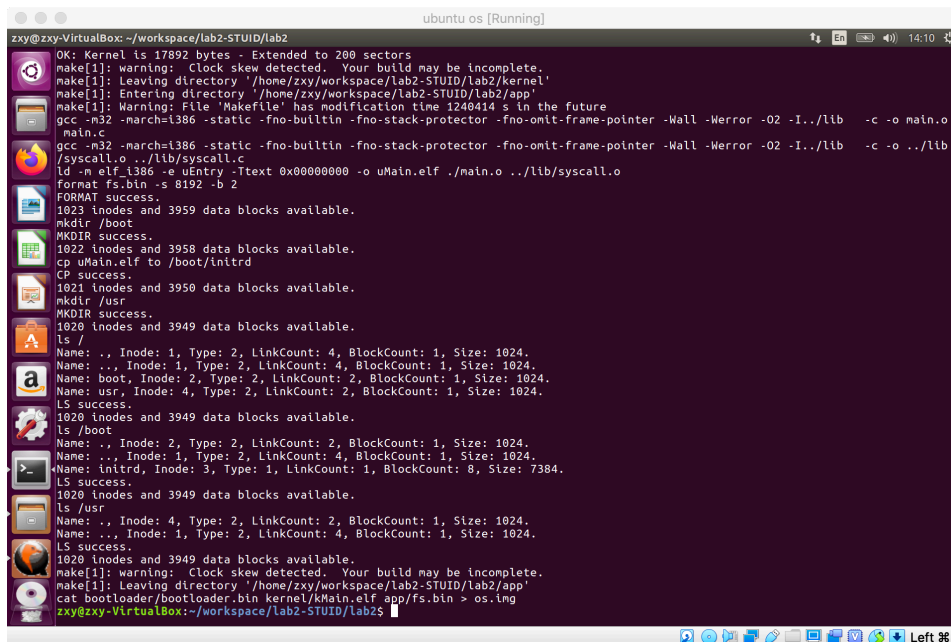
实验二 实验报告

一、实验进度

我完成了格式化程序，完成cp命令；键盘按键的串口回显；实现系统调用库函数printf和对应的处理例程以及printf的格式化输出。

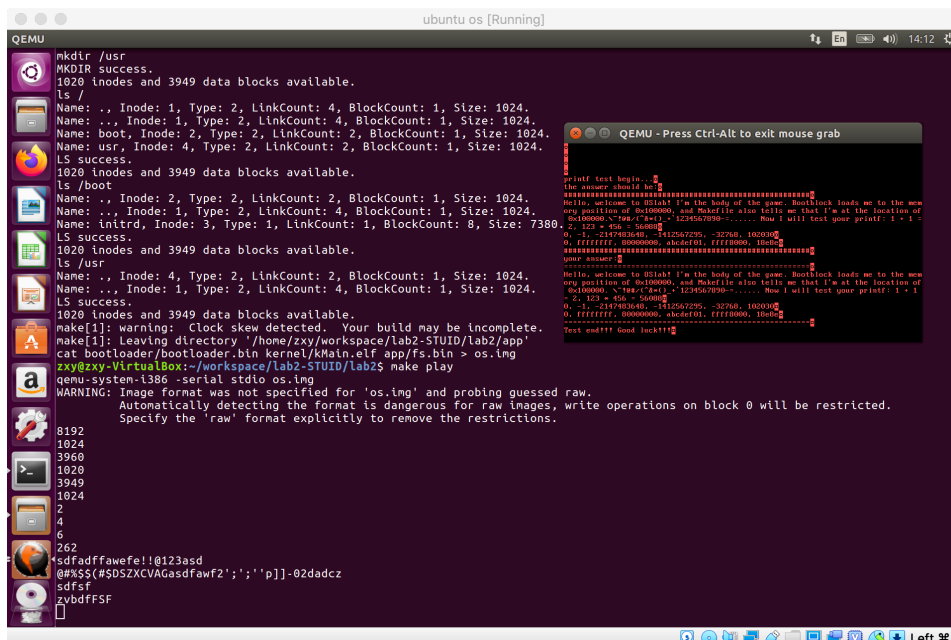
二、实验结果

1. 格式化程序：



```
zxy@zxy-VirtualBox: ~/workspace/lab2-STUID/lab2
OK: Kernel is 17892 bytes - Extended to 288 sectors
make[1]: warning: Clock skew detected. Your build may be incomplete.
make[1]: Leaving directory '/home/zxy/workspace/lab2-STUID/lab2/kernel'
make[1]: Entering directory '/home/zxy/workspace/lab2-STUID/lab2/app'
make[1]: Warning: File 'Makefile' has modification time 1240414 s in the future
gcc -m32 -march=i386 -static -fno-builtin -fno-stack-protector -fno-omit-frame-pointer -Wall -Werror -O2 -I../lib -c -o main.o main.c
gcc -m32 -march=i386 -static -fno-builtin -fno-stack-protector -fno-omit-frame-pointer -Wall -Werror -O2 -I../lib -c -o ../lib
/syscall.o ../lib/syscall.c
ld -m elf_i386 -e uEntry -Ttext 0x00000000 -o uMain.elf ./main.o ../lib/syscall.o
format fs.bin -s 8192 -b 2
FORMAT success.
1023 inodes and 3959 data blocks available.
mkdir /boot
MKDIR success.
1022 inodes and 3958 data blocks available.
cp uMain.elf to /boot/initrd
CP success.
1021 inodes and 3950 data blocks available.
mkdir /usr
MKDIR success.
1020 inodes and 3949 data blocks available.
ls /
Name: ., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: boot, Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: usr, Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
LS success.
1020 inodes and 3949 data blocks available.
ls /boot
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 8, Size: 7384.
LS success.
1020 inodes and 3949 data blocks available.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
LS success.
1020 inodes and 3949 data blocks available.
make[1]: warning: Clock skew detected. Your build may be incomplete.
make[1]: Leaving directory '/home/zxy/workspace/lab2-STUID/lab2/app'
cat bootloader/bootloader.bin kernel/kMain.elf app/fs.bin > os.img
zxy@zxy-VirtualBox:~/workspace/lab2-STUID/lab2$
```

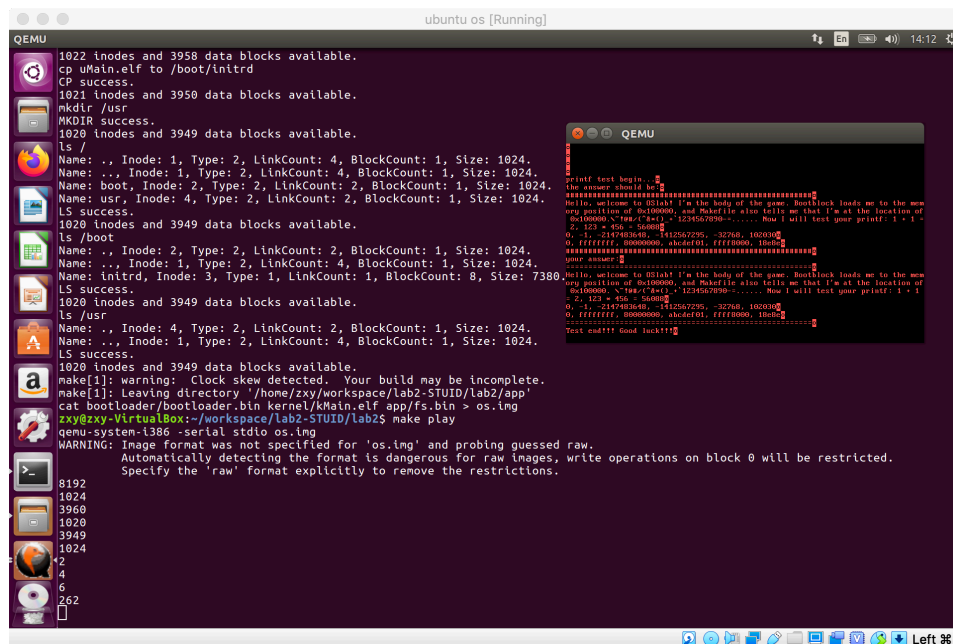
2. 键盘按键的串口回显：



```
QEMU
mkdir /usr
MKDIR success.
1020 inodes and 3949 data blocks available.
ls /
Name: ., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: boot, Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: usr, Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
LS success.
1020 inodes and 3949 data blocks available.
ls /boot
Name: ., Inode: 2, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
Name: initrd, Inode: 3, Type: 1, LinkCount: 1, BlockCount: 8, Size: 7380.
LS success.
1020 inodes and 3949 data blocks available.
ls /usr
Name: ., Inode: 4, Type: 2, LinkCount: 2, BlockCount: 1, Size: 1024.
Name: .., Inode: 1, Type: 2, LinkCount: 4, BlockCount: 1, Size: 1024.
LS success.
1020 inodes and 3949 data blocks available.
make[1]: warning: Clock skew detected. Your build may be incomplete.
make[1]: Leaving directory '/home/zxy/workspace/lab2-STUID/lab2/app'
cat bootloader/bootloader.bin kernel/kMain.elf app/fs.bin > os.img
zxy@zxy-VirtualBox:~/workspace/lab2-STUID/lab2$ make play
qemu-system-i386 -serial stdio os.img
WARNING: Image format was not specified for 'os.img' and probing guessed raw.
Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
Specify the 'raw' format explicitly to remove the restrictions.

8192
1024
3960
1020
3949
1024
2
4
6
262
'sdfadffawefe!!@123asd
@#f$S($SD$ZXC$VAGasdfawF2';';'p]]-02dadcz
sdfsf
2VbdfFSF
[]
```

3. 实现系统调用库函数printf和对应的处理例程：



三、实验代码

1. utils/genFS/func.c中

完成cp命令，使得宿主机上的文件可以拷贝到boot/initrd中。

完成代码可参考touch命令的实现，通过减少对filepath的合法判定约束和copydata函数来实现cp命令。

2. kernel/kernel/idt.c 中 void initIdt() 函数

设置门描述符

kernel/kernel/irqHandle.c 中 void keyboardHandle() 函数

利用键盘驱动getKeyCode()和putChar()完善中断服务例程。

3. kernel/kernel/irqHandle.c 中 void syscallPrint() 函数

通过写显存实现printf的处理例程。注意‘\n’与输输出到行末两种换行的情况，以及最后一行换行情况需要scrollScreen。

4. lib/syscall.c 中 void printf(const char *format,...)函数

完善printf的格式化输出。将%d %x %s %c四种格式通过转化运用参数，转化为字符串最后存储在buffer。

注意通过判断 '%' 与之后格式进行参数的代入与状态机的转换，以及缓冲区达到最大后的系统服务例程调用。

四、实验中遇到的问题以及解决方法

实现printf时，format所在地址应该指向printf的第一个字符串参数。所以在索引format之后printf中的参数时，index初值应设为1。

五、思考题

问题一、分别考虑内存分段机制，**ext**文件系统和内存分页机制，它们之间是否有某种联系？回忆**i386**分页机制，为什么**ext**文件系统没有采用类似于分页机制中对物理页的组织方式，通过固定的两级索引来寻找物理页？

答：1. 联系：都是将空间划分成一定大小的单位，再通过检索来组织管理空间。通过这种方式，来提高较小碎片空间单位的利用效率。2. 原因：分页机制是对于内存的管理机制，内存的存储空间较大且固定大小，用固定的两级索引效率更高。ext文件系统，每个文件的大小都是不相同的。当文件较小时只需要使用前几个index直接索引，当文件较大时再引入二级、三级索引块。这样针对不同大小文件效率更高。

问题二、你知道下表中的数据是如何计算出来的吗？

块大小	1KB	2KB	4KB
一个文件最大大小	16GB	256GB	2TB
文件系统最大大小	4TB	8TB	16TB

答：

文件最大大小的理论值计算：

块	块中指针数量	数据块数量	文件大小
1KB	256	$11+256+256^2+256^3 = 16843019$	16GB
2KB	512	$11+512+512^2+512^3 = 134480395$	256GB
4KB	1024	$11+2^{10}+2^{20}+2^{30} = 1074791435$	4TB

因为使用4byte的块指针，所以最多可以指向 2^{32} 个数据块，文件系统最大大小理论值计算：

块	1KB	2KB	4KB
文件系统大小	$1\text{KB} * 2^{32} = 4\text{TB}$	$2\text{KB} * 2^{32} = 8\text{TB}$	$4\text{KB} * 2^{32} = 16\text{TB}$

问题三、在磁盘上创建一个新文本文件，往里面写入一个字符后，查看该文件的大小：

- 在Windows中，有一项叫**Size on disk**的数据
- 在Linux中，使用 **ls -ls** 命令，输出的第一列即为文件占用的磁盘空间（单位：**KB**） 这些显示的数值远远大于一个字节，你知道为什么吗？

答：文件分为文件大小和占用空间大小。大小表示文件实际数据的字节数；占用空间大小表示文件占用硬盘空间大小的字节数。在Linux系统中，一个字符将保存在一个块中，并由文件的inode索引，此时文件占用磁盘空间为一个block的大小。

问题四、Linux不允许对目录创建硬链接，你知道为什么吗？如果允许对目录创建硬链接，你能否举出一个影响系统工作的例子？

答：因为Linux文件系统目录隐藏了两个特殊的目录：当前目录（.）和父目录（..）。这特殊目录的inode号显示出这两个目录就是两个硬链接。若系统允许对目录创建硬链接，则会产生目录环。

举例：

```

father
|- .
|- ..
|- son
|- .
|- ..

```

此时将father与son目录建立硬连接，则father与son目录使用相同的inode及data block。son目录的父目录为father，此时son目录与父目录建立了硬链接，想从son目录返回上级目录则陷入死循环。

问题五、IA-32提供了4个特权级，但TSS中只有3个堆栈位置信息，分别用于ring0, ring, ring2的堆栈切换。为什么TSS中没有ring3的堆栈信息？

答：程序在外层切换到内层过程中寻找内层堆栈信息时才需要用到TSS，因此最外层ring3的信息不用在此保留。

问题六、我们在使用eax, ecx, edx, ebx, esi, edi前将寄存器的值保存到了栈中，如果去掉保存和恢复的步骤，从内核返回之后会不会产生不可恢复的错误？

答：在内核执行程序时，修改了寄存器的值，从内核返回后寄存器的值发生了改变。在通过寄存器的值进行地址读取时将会发生未知的错误。

六、实验心得

通过对代码结构的理解，读懂每一个程序的含义及其起到的作用是顺利完成实验的关键。实验中需要对实验指南进行研读，并能积极的对一些功能进行实践，保持良好学习心态。这是健康的学习成长经历。