第一章 Linux 系统知识

1、开发环境？

Ubuntu+XSHELL+XFTP+VScode

XSHELL：Windows 端远程操作 Linux 端工具。需要 Linux 服务器名称和 IP 地址（Linux：sudo apt install openssh-server. 默认开启服务。Ifconfig 查看 ens33IP）。

VScode：添加 remote development 插件，远程链接到 Linux；远程资源扩展器中选择 SSH，并选择配置文件\config，添加用户名和 IP 地址（建立远程连接）；管理→命令面板：C\C++编辑配置，在创建的.vscode 目录下生成的 c_cpp_properties.json 文件中的 includePath 字段中添加 “/usr/include”（用于解决显示找不到头文件）；在扩展中安装 C\C++插件；创建和修改 launch.json 文件：①运行→创建文档→选择 C++→选择默认；②运行→选择 gdb 启动→鼠标右键配置调试信息。修改 miDebuggePath 指定调试器，preLaucchTask 编译代码生成的可执行文件；创建 task.json 文件，终端→配置任务→使用模板→Others，修改 label 和 command：gcc，添加 args 参数。顺序：创建文件-Json-launch.json-task.json（创建的文件不能被包含在./vscode 里面！）。

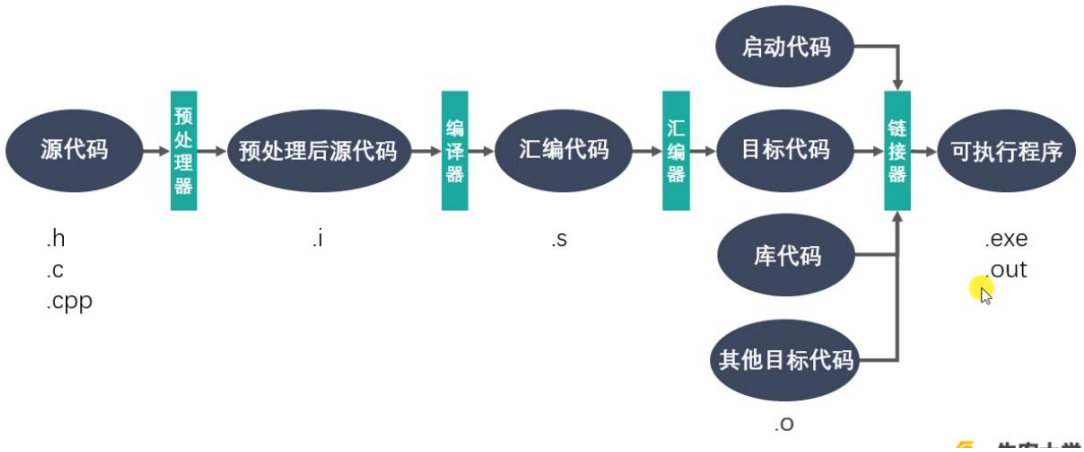
2、GCC（GNU Compiler Collection）？

安装命令：sudo apt install gcc g++

常用命令：mkdir program 创建目录；touch test.c 创建文件；

vim test.c 编辑文件；gcc test.c –o app；gcc test.c 编译，rm test.c 删除文件。

GCC 工作流程：



gcc test.c –E –o test.i；gcc test.i –S –o test.s；gcc test.s –s –o test.o；

3、GCC 和 G++的区别？

.c 文件：gcc 当作 c 文件；g++当作 c++。

.cpp 文件：都是 c++。

编译阶段：gcc 不能自动和 c++程序使用的库联接，所以用 g++。

4、库文件？

库是一种不能单独运行的程序，可以提供给用户直接使用的变量函数或类。

静态库和动态库：静态库在程序链接阶段被复制到程序中；动态库在程序运行时由系统动态加载到内存中调用。

库的优点：代码保密；方便部署和开发。

5、静态库？

命名规则：Linux：libxxx.a　　Windows：libxxx.lib

lib：固定前缀

xxx：自定义库名字

.a：固定后缀

静态库制作（Linux）：

gcc 获得 .o 文件；将 .o 文件打包，使用 ar 工具（archive）

ar rcs libxxx.a xxx.o xxx.o

r ：将文件插入备存文件中 c：建立备存文件 s：索引。

静态库使用（Linux）：

gcc test.c –o test –I(大写 i) ./include/(头文件位置) –lcalc(静态库的名字) -L ./lib(库位置)。

6、动态库？

命名规则：Linux：libxxx.so 可执行文件。Windows：libxxx.dll。

动态库制作：

gcc 得到 .o 文件，位置无关！gcc –c –fpic/-fPIC a.c b.c。

gcc –shared a.o b.o –o libcacl.so。

使用报错：找不到动态库文件？

解析：程序运行时，动态库会被加载到内存中，通过 ldd（list dynamic dependencies）命令检查动态库依赖关系。 命令：ldd 可

运行程序名称。如何定位动态库文件呢？当系统加载可执行代码时，能够知道依赖库的名字，但是还需要知道绝对路径。需要通过动态载入器来获取绝对路径。对于 elf 格式的可执行程序，由 ld-linux.so 来完成，先后搜索 elf 文件的 DT_RPATH 段 → 环境变量 LD_LIBRARY_PATH → /etc/ld.so.cache 文件列表 → /lib/, /usr/lib 目录找到库文件后将其加载到内存。

解决：①在终端中临时配置：

env：查看环境变量（shell 命令是通过环境变量去实现的)。

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH(获取原来的值):/home/lib（动态库目录地址)：添加环境变量（目的是把动态库文件添加到环境变量中，pwd 显示当前目录)。

echo $LD_LIBRARY_PATH：打印 LD_L 值的目录地址。

②永久配置：

用户级配置：cd 进入 home 目录；vim .bashrc：export；. .bashrc（. ./.bashrc；source .bashrc)。

系统级配置：sudo vim /etc/profile；export；source /etc/profile。

③修改/etc/ld.so.cache 文件：sudo vim /etc/ld.so.config；复制路径；sudo ldconfig。

④修改/lib/,/usr/lib/目录：不建议：因为有系统文件。

7、Makefile?

工程的源文件很多，按类型、功能、模块分别放在若干目录中。

Makefile 文件定义了一系列的规则来指定文件的编译顺序、是否重新编译等功能。好处就是自动化编译，只需要一个 make 命令，整个工程就可以完全自动化编译。

规则：

- Makefile 规则
  - 一个 Makefile 文件中可以有一个或者多个规则

```
目标 ...: 依赖 ...
    命令 (Shell 命令)
    ...
```

  - 目标：最终要生成的文件（伪目标除外）
  - 依赖：生成目标所需要的文件或是目标
  - 命令：通过执行命令对依赖操作生成目标（命令前必须 Tab 缩进）

```
app:sub.o add.o mult.o div.o main.o
        gcc sub.o add.o mult.o div.o main.o -o app

sub.o:sub.c
        gcc -c sub.c -o sub.o

add.o:add.c
        gcc -c add.c -o add.o

mult.o:mult.c
        gcc -c mult.c -o mult.o

div.o:div.c
        gcc -c div.c -o div.o

main.o:main.c
        gcc -c main.c -o main.o
```

04 / 变量

- 自定义变量

  变量名=变量值　　var=hello　　$(var)

- 预定义变量

  AR ： 归档维护程序的名称，默认值为 ar

  CC ： C 编译器的名称，默认值为 cc

  CXX ： C++ 编译器的名称，默认值为 g++

  $@ ： 目标的完整名称

  $< ： 第一个依赖文件的名称

  $^ ： 所有的依赖文件

- 获取变量的值

  $(变量名)

```
app:main.c a.c b.c
    gcc -c main.c a.c b.c
```

```
#自动变量只能在规则的命令中使用
app:main.c a.c b.c
    $(CC) -c $^ -o $@
```

05 / 模式匹配

```
add.o:add.c
    gcc -c add.c
div.o:div.c
    gcc -c div.c
sub.o:sub.c
    gcc -c sub.c
mult.o:mult.c
    gcc -c mult.c
main.o:main.c
    gcc -c main.c
```

```
%.o:%.c
```

- %：通配符，匹配一个字符串
- 两个%匹配的是同一个字符串

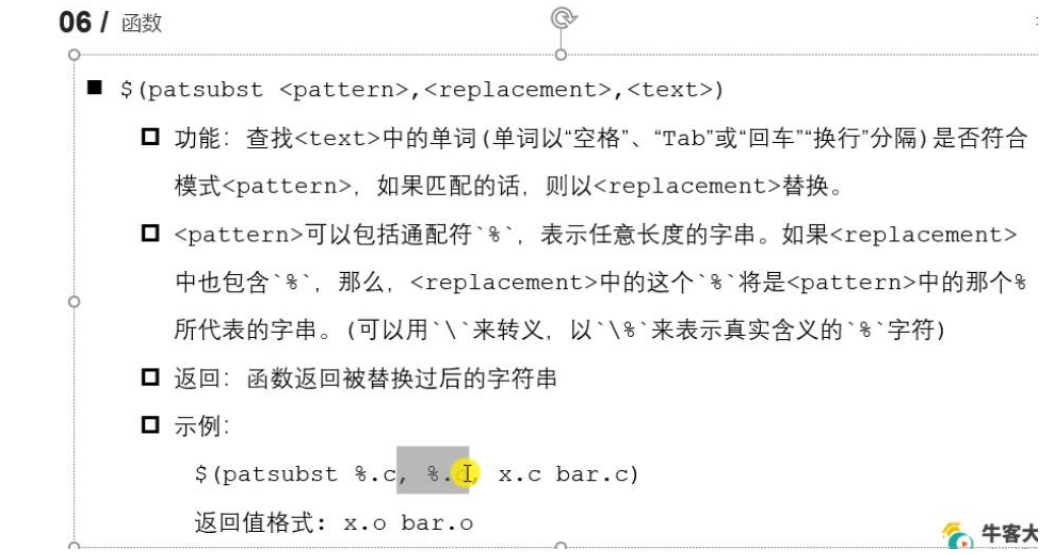
```
%.o:%.c
    gcc -c $< -o $@
```

06 / 函数

- $(wildcard PATTERN...)
  - 功能：获取指定目录下指定类型的文件列表
  - 参数：PATTERN 指的是某个或多个目录下的对应的某种类型的文件，如果有多个目录，一般使用空格间隔
  - 返回：得到的若干个文件的文件列表，文件名之间使用空格间隔
  - 示例：

    $(wildcard *.c ./sub/*.c)

    返回值格式：a.c b.c c.c d.c e.c f.c




```
#定义变量
# add.c sub.c main.c mult.c div.c
src=$(wildcard ./*.c)
objs=$(patsubst %.c, %.o, $(src))
target=app
$(target):$(objs)
        $(CC) $(objs) -o $(target)

%.o:%.c
        $(CC) -c $< -o $@
~
~
~
```

Makefile 文件删除*.o 文件：

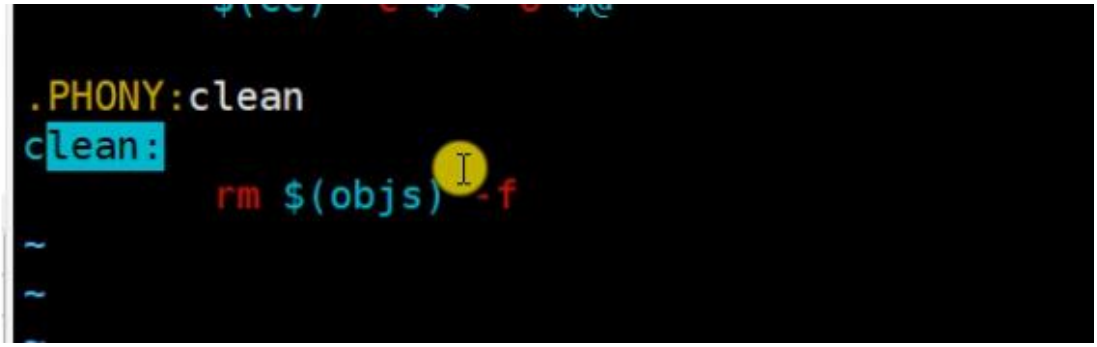
```
%.o:%.c
        $(CC) -c $< -o $@

clean:
        rm $(objs) -f
```

然后执行命令：make clean（直接执行 make 会提醒可执行

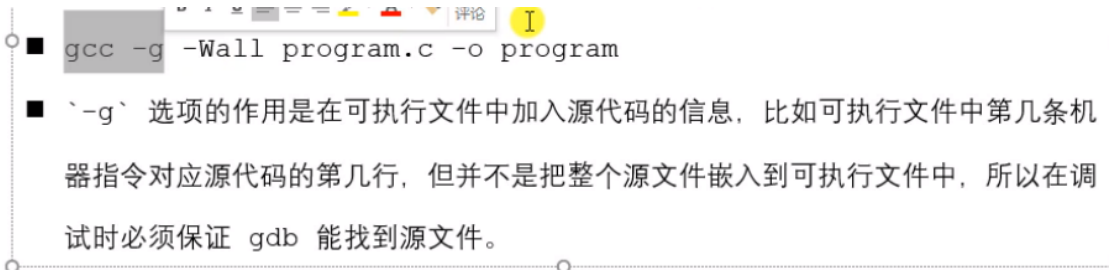
程序时最新的，因为没有改写任何的.c 文件)。-f：不提示强制删除。

可以将 clean 设置成伪目标，从而不会生成目标 clean 文件。

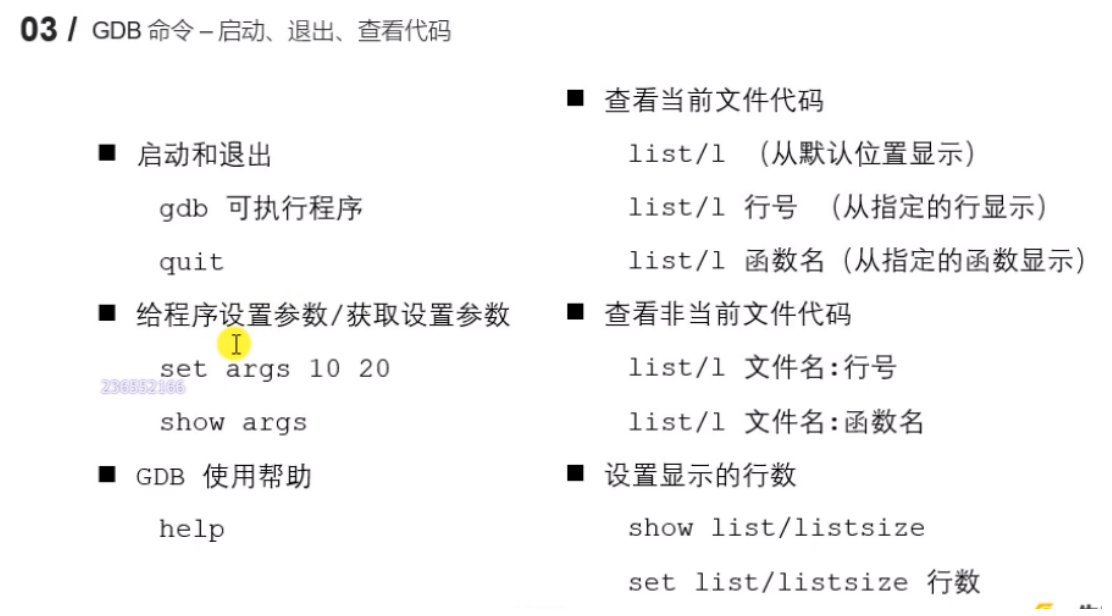


8、GDB?

GDB 是 GNU 提供的调试工具。自定义运行程序；断点处停止。

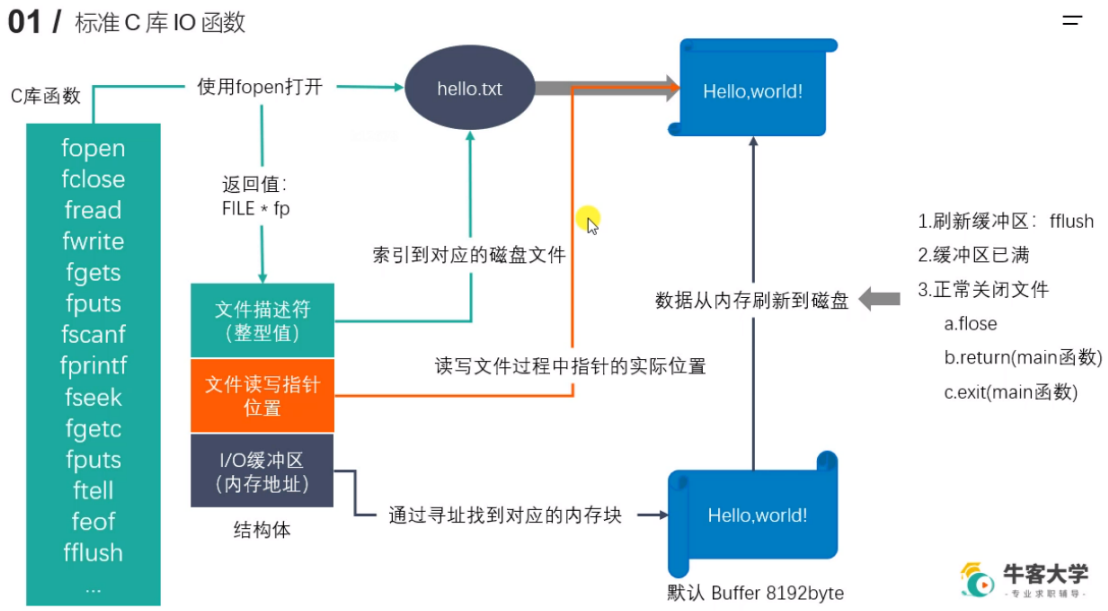


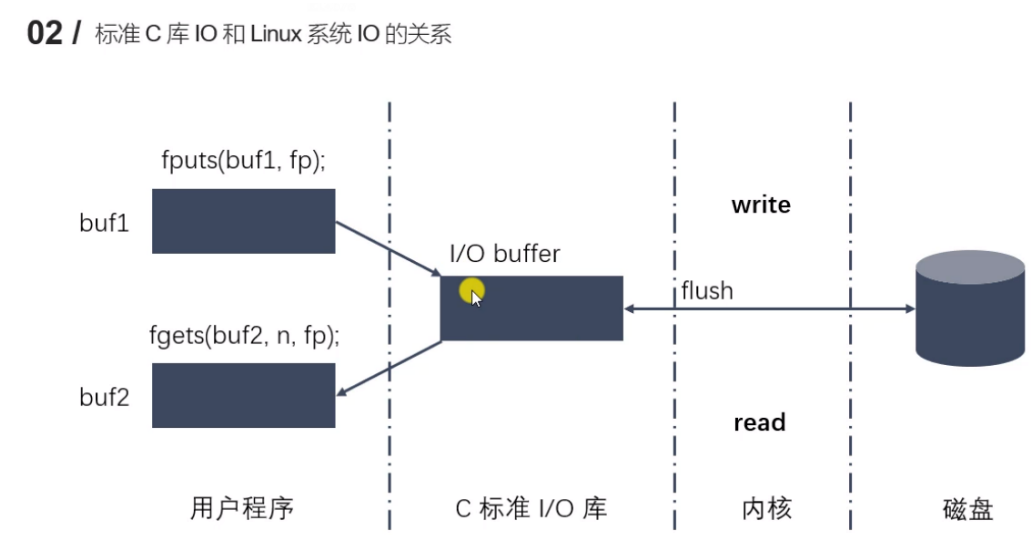
gcc test.c –g –o test。



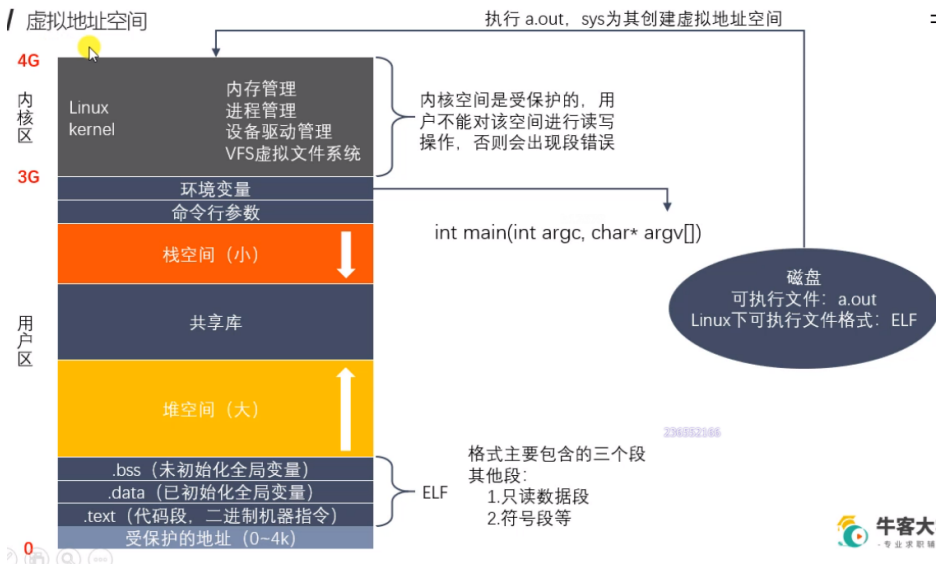
9、IO?

Linux 的 IO 函数没有缓冲区，读写速度要比含有缓冲区的标准 C 库 IO 函数要慢，调用一次 read 函数就要读一次磁盘。在网络通信领域，要追求效率，所以使用 Linux 的 IO 函数去读写。在磁盘读写的操作，需要缓冲区提高读写效率，选择标准 C 库 IO 函数。

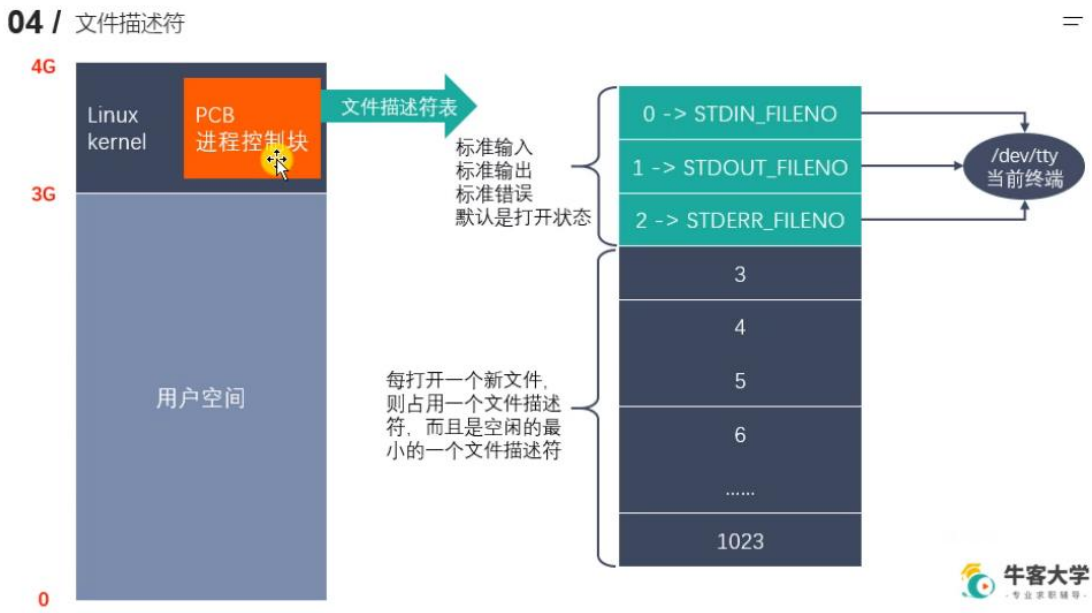

01 / 标准 C 库 IO 函数
C库函数
使用fopen打开
hello.txt
Hello,world!
fopen
fclose
fread
fwrite
fgets
fputs
fscanf
fprintf
fseek
fgetc
fputs
ftell
feof
fflush
...
返回值:
FILE * fp
索引到对应的磁盘文件
文件描述符
（整型值）
文件读写指针
位置
I/O缓冲区
（内存地址）
结构体
读写文件过程中指针的实际位置
数据从内存刷新到磁盘
1.刷新缓冲区：fflush
2.缓冲区已满
3.正常关闭文件
a.flose
b.return(main函数)
c.exit(main函数)
通过寻址找到对应的内存块
Hello,world!
默认 Buffer 8192byte
牛客大学


02 / 标准 C 库 IO 和 Linux 系统 IO 的关系
fputs(buf1, fp);
buf1
write
I/O buffer
flush
fgets(buf2, n, fp);
buf2
read
用户程序
C 标准 I/O 库
内核
磁盘

10、虚拟地址空间?


/ 虚拟地址空间
执行 a.out，sys为其创建虚拟地址空间
4G
内核区
Linux kernel
内存管理
进程管理
设备驱动管理
VFS虚拟文件系统
内核空间是受保护的，用户不能对该空间进行读写操作，否则会出现段错误
3G
环境变量
命令行参数
int main(int argc, char* argv[])
栈空间（小）
共享库
用户区
堆空间（大）
磁盘
可执行文件：a.out
Linux下可执行文件格式：ELF
.bss（未初始化全局变量）
.data（已初始化全局变量）
.text（代码段，二进制机器指令）
受保护的地址（0~4k）
ELF
格式主要包含的三个段
其他段:
1.只读数据段
2.符号段等
0
牛客大

11、文件描述符?



12、Linux IO 函数? （man 2 查看系统内核常调用函数）

open：(man 2 open)打开文件/创建新文件

```
nana@nana-virtual-machine:~/linux$ man 2 open
nana@nana-virtual-machine:~/linux$ man 3 perro
```

```
nana@nana-virtual-machine:~/linux$ umask
0002
```

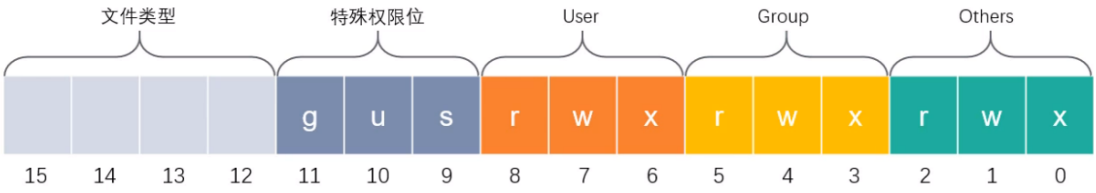
```
nana@nana-virtual-machine:~/linux$ ls
test.c
nana@nana-virtual-machine:~/linux$ gcc test.c -o test
nana@nana-virtual-machine:~/linux$ ls
test  test.c
nana@nana-virtual-machine:~/linux$ ./test
open: No such file or directory
nana@nana-virtual-machine:~/linux$ ls
create.txt  test  test.c
nana@nana-virtual-machine:~/linux$ ll create.txt
-rwxrwxr-x 1 nana nana 0  9月  9 17:39 create.txt*
```

详见 Linux 系统下 Linux 文件夹。

stat 函数：

07 / st_mode 变量

setGID – 设置组id
setUID – 设置用户id
Sticky – 粘住位



- S_IFSOCK 0140000 套接字
- S_IFLNK 0120000 符号链接（软链接）
- S_IFREG 0100000 普通文件
- S_IFBLK 0060000 块设备
- S_IFDIR 0040000 目录
- S_IFCHR 0020000 字符设备
- S_IFIFO 0010000 管道
- S_IFMT 0170000 掩码

(st_mode & S_IFMT) == S_IFREG

- S_IRUSR 00400
- S_IWUSR 00200
- S_IXUSR 00100
- S_IRWXU 00700

- S_IRGRP 00040
- S_IWGRP 00020
- S_IXGRP 00010
- S_IRWXG 00070

- S_IROTH 00004
- S_IWOTH 00002
- S_IXOTH 00001
- S_IRWXO 00007