VIM:

12种模式:6种基本模式,6种条件模式

vim [文件名] 进入文件

①普通模式🡪插入模式

1. i进入插入模式 光标在当前位置
2. I进入插入模式 光标到行首
3. a光标移动到下一个字符处开始插入
4. A 光标移到行尾开始插入
5. o在光标所在行下方添加新行并进入插入模式
6. O在光标所在行上方添加新行并进入插入模式
7. s删除光标所在字母并进行插入

插入模式->普通模式

Esc键

②普通模式下的操作:

h 左

j 下

k 上

l 右

yy 复制行

nyy 从光标所在行开始复制n行

p 粘贴到光标所在行的下方

dd 剪切光标所在行

ndd 从光标所在行开始删除(剪切)N行数据

w 光标按单词向右移动

b 光标按单词向左移动

dw 从光标所在位置删除单词

x 删除光标所在单词

u 撤销上一次操作

ctrl+r 反向撤销

ctrl+v 进入列视图选中模式

v 进入行视图选中模式

r 替换

ctrl+v进入视图模式 hjkl选中 = 按C语言文件风格缩进对齐

gg=G 选中全部并整理格式

gg 光标移到到文件首行

G 光标移动到文件尾行

:w 保存

:q 退出

:wq 保存提出

:q! 退出不保存

:[命令] 执行命令 回车回到VIM

gcc工具使用:

gcc将一个高级语言编译成机器可识别程序经历的过程:

1.预处理

展开所有代码

2.编译

纠错-🡪将程序编译为汇编代码

3.汇编

将汇编代码编译成为机器可识别的指令代码

前边三步只是将自身代码编译成为编译机器代码,但在我们的代码中有很多调用函数不是我们实现的,但是如果我们要生成最终的可执行程序 那么我们就必须在我们的代码中能够知道这些函数的实现.

操作系统提供的方法:系统调用接口

系统调用不太好用,因此有一些大拿就对接口进行了封装,最终打包出一个库,因此我们在使用这些函数时,包含头文件 并且链接库即可.

4.链接

生成可执行文件或库文件

就是将别人已经实现的代码拿过来,跟我们的程序一起生成最终的可执行程序.

动态链接:记录函数地址信息,并不是将代码全拿过来,所以可执行文件较小,但是运行时依赖库

静态链接:将库中的代码全部拿过来,跟我们的程序一起生成可执行程序,最终生成的可执行程序较大,但是运行的时候不需要依赖库

gcc默认是动态链接

gcc -E [文件] 只预处理 后停止编译

gcc -S[文件] 只编译

gcc -c[文件] 只汇编

gcc -o [文件] 指定生成文件名称

yum工具使用:

yum install 包名 安装软件包

lrzsz 文件传输工具

yum remove 包名 卸载软件包

yum list 列出可安装的软件包

yum list | grep lrzsz 因可列的软件包太多所以用管道符过滤

Makefile

是不普通的普通文件

操作系统中有一个命令叫make

makefile中记录的是一系列的编译规则,当我们执行make命令 那么make这个命令就会到当前目录下查找makefile文件 然后一步步解释执行makefile中记录的编译规则,最终通过这些编译规则,完成项目编译.

makefile中#开头是注释

makefile的编写规则:

|生成的对象: 依赖对象

|(tab键) 为了完成生成此生成对象的生成所要执行的命令

eg:

main: main.c

gcc main.c -o main

makefile编写完毕,保存退出后,直接敲击make命令,那么make命令所对应的程序会到当前目录下找到makefile文件,然后按照makefile中记录的编译规则来完成最终项目的编译

因此makefile文件名字只能叫makefile/Makefile

makefile会记录源文件和程序文件的时间信息,如果源文件的最后一次修改时间是在程序文件之后,那么就会重新编译程序,否则将报告程序是最新的,而不需要重新编译

make永远只生成第一个对象,执行完为为生成对象所要执行的命令后就退出,不再继续生成第二个对象

如果makefile有多个要生成的对象,那么就需要在最上边定义一个对象,将要生成的对象都作为这个对象的依赖对象

eg:

all:main child

main:main.c

gcc main.c -o main

child:child.c

gcc child.c -o child

声明伪对象,伪对象代表每次都可以重新生成:

**.PHONY:[对象] [对象] ....**

执行完文件想删除文件,就可以在makefile中写

clean:

rm -f [文件] [文件]

但不能把clean写进all里;

此时make可以在外部指定删除生成的对象 make clean

makefile中的预定义变量:

makefile中$符号用于使用变量的值

eg:

a=1

a=$b

$@ 冒号左边的要生成对象

$< 冒号右边的依赖对象中的第一个

$^ 冒号右边的所有依赖对象

main:main.c

gcc $^ -o $@

eg:

main:main.c

gcc main.c -o main

wildcard 获取所有文件名

% 通配符

patsubst 对后者的变量中的内容进行替换

万能makefile文件

获取当前目录下所有的.c文件放到SRC变量中

SRC=$(wildcad ./\*.c)

SRC变量中的内容进行替换,将.c后缀替换为.o后缀赋值给OBJ变量

OBJ=$(patsubst %.c,%.o,$(SRC))

将所有的.o文件链接最终生成可执行程序

BIN=./main

将所有.c文件只是进行了汇编生成.o文件

%.o:%.c

gcc -c $^ -o $@

.PHONY:clean

clean:

rm -f $(OBJ) $(BIN)

make -c [dir] 进入指定目录下执行make命令

课后调研:makefile中变量赋值有好几种方式, 这四种变量赋值有什么不同???

= 变量的值不是在声明而是在使用时递归展开

:= 直接使用value的值来设置来变量 - 它的值在声明时就展开了

?= 只有在变量未被设置时设置成value

+= 把右边值追加到左边上（或是如果变量不存在则设置该值）

gdb的使用

gdb是linux下最火的程序调试工具

如何运行程序 gdb ./main

调试正在运行的程序 gdb -p [进程id]

获取进程id ps -ef |grep [文件]

程序后边不需要参数信息，因为参数的获取这里无效,这里仅加载可执行程序信息

进入gdb调试后的命令操作： run/r 运行程序，后跟程序的运行参数r -l/r -a/r -i

开始逐步调试 start

默认查看当前位置附近五行的代码 liist/l

逐步调试代码(跳过函数) n

继续运行余下所有代码 c

给指定文件的指定行加断点 break [文件]:行数

给指定文件的指定函数加断点 b [函数名]

查看断点 info break

查看信息 info

默认删除所有断点 d

根据断点id删除指定断点 d bid

逐步跟踪进入函数 step

打印变量 p [变量]

变量监控 watch [变量]

查看函数调用栈信息（出错时） bt