

实验题目：lab0，操作系统的编程基础

171491125 吴俊

安装 linux 环境，并安装 gcc 和 gdb。

1. 了解汇编

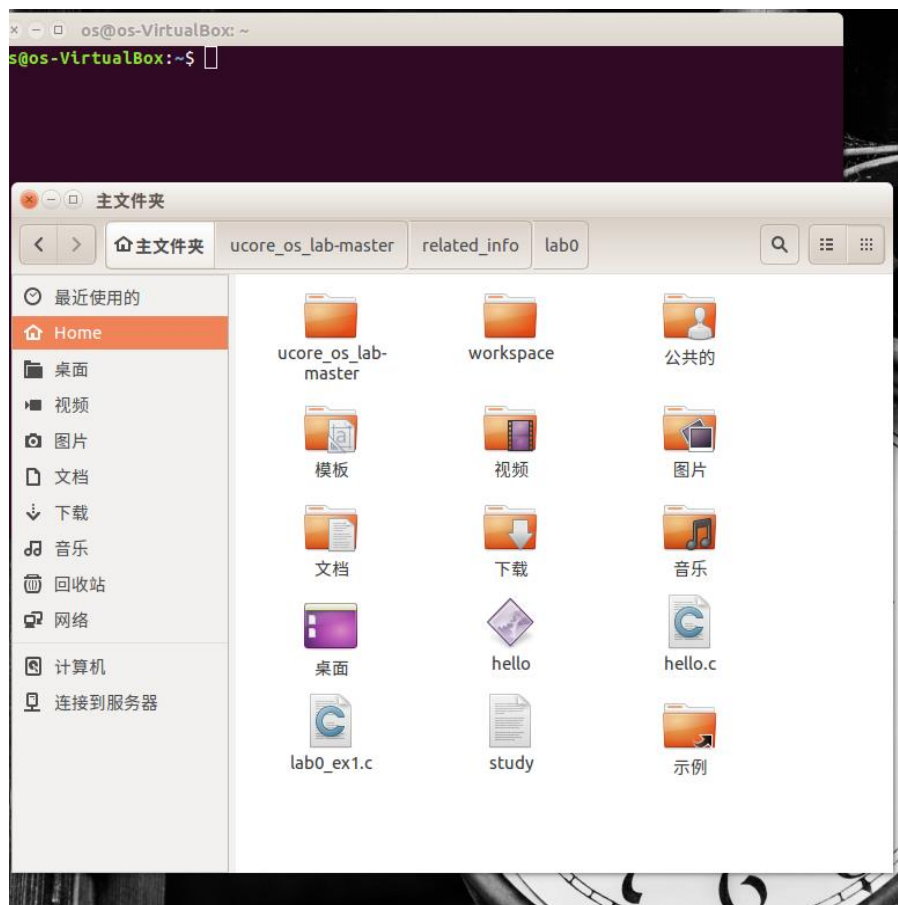
尝试理解下面的命令

```
$gcc -S -m32 lab0_ex1.c
```

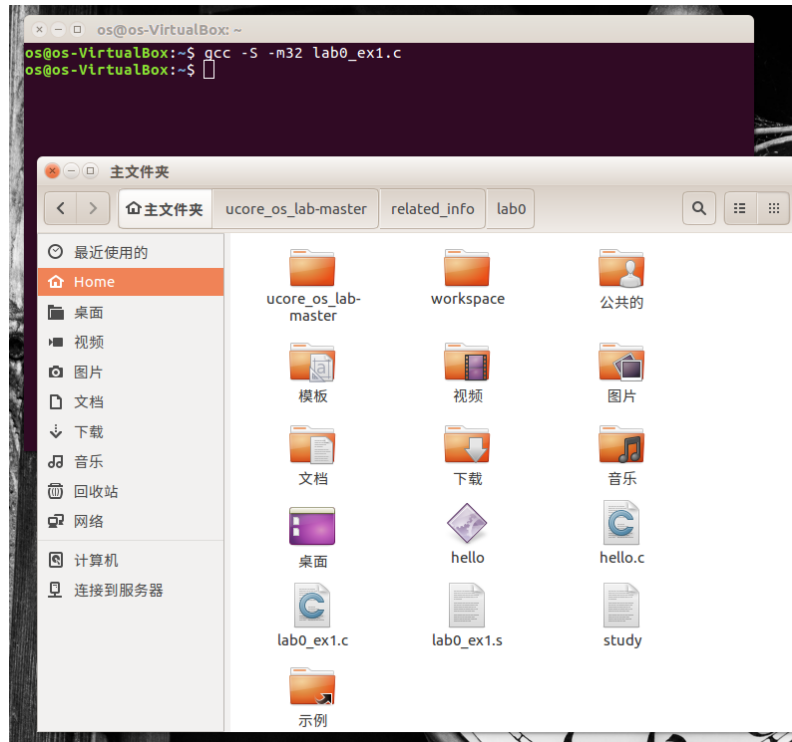
接着我们将得到 lab0_ex1.s 文件，请写出汇编代码与 c 代码之间的关系。

Lab0_ex1.c 这个文件，原本不在 home 根目录里。所以，我试了这个指令好多次都找不到这个文件。后来，我把这个文件放到了 home 根目录里，就可以成功找到，并且编译了后缀为.s 的文件。如果找到 Lab0_ex1.c 所在的位置，右键，在终端打开也可以。

输入命令之前：



输入命令之后：



汇编代码与 C 代码之间的关系：

C 代码经过 gcc 的编译（仅编译），得到汇编代码。

```
-S Compile only; do not assemble or link
```

通过使用命令 `gcc -help`，更加验证了我的猜想。

2. 用 gdb 调试

尝试下面的命令，

```
$gcc -g -m32 lab0_ex2.c
```

接着我们会得到 `a.out` 文件，请用 `gdb` 调试，并写出设置断点、单步执行及查看变量的过程。

1. 同上，在使用此命令时，也要把 `lab0_ex2.c` 这个文件移到 `home` 下。否则会显示找不到 `lab0_ex2.c` 这个文件或目录。

2. 如果不在上述命令后使用 `-o` 这个选项，得到的文件均为 `a.out`。

3. 在使用 `gdb` 调试时，首先要打开自己的 C 代码，根据 C 代码进行调试。

- 首先要把需要调试的可执行文件装入。命令：`gdb a.out`
- 如果此步骤之后输入 `r` 便直接运行这个程序了，但是一般不这样，都是设置断点。

设置断点方法：例，在第三行设置断点，命令 b 3。

- 接着，gdb 将在“n”处设置观察点。输入 n 后，单步运行程序。
- 如果要查看变量，一般都是通过将变量打印出来查看。所以仅需输入命令 p 【变量名】，打印基本类型，数组，字符数组。如果有时候，多个函数或者多个文件会有一个变量名，这个时候可以加上文件名或者函数名来区分。

装入文件

```
os@os-VirtualBox: ~  
os@os-VirtualBox:~$ gdb a.out  
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1  
Copyright (C) 2016 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law. Type "show copying"  
and "show warranty" for details.  
This GDB was configured as "i686-linux-gnu".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
<http://www.gnu.org/software/gdb/bugs/>.  
Find the GDB manual and other documentation resources online at:  
<http://www.gnu.org/software/gdb/documentation/>.  
For help, type "help".  
Type "apropos word" to search for commands related to "word"...  
Reading symbols from a.out...done.  
(gdb)
```

设置断点

```
os@os-VirtualBox: ~  
os@os-VirtualBox:~$ gdb a.out  
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1  
Copyright (C) 2016 Free Software Foundation, Inc.  
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law. Type "show copying"  
and "show warranty" for details.  
This GDB was configured as "i686-linux-gnu".  
Type "show configuration" for configuration details.  
For bug reporting instructions, please see:  
<http://www.gnu.org/software/gdb/bugs/>.  
Find the GDB manual and other documentation resources online at:  
<http://www.gnu.org/software/gdb/documentation/>.  
For help, type "help".  
Type "apropos word" to search for commands related to "word"...  
Reading symbols from a.out...done.  
(gdb) b 3  
Breakpoint 1 at 0x536: file lab0_ex2.c, line 3.  
(gdb)
```

单步执行：

```
(gdb) n  
Hello, world!  
6      in lab0_ex2.c  
(gdb)
```

查看变量：

```
Hello, world!  
6      in lab0_ex2.c  
(gdb) p main  
$1 = {int (void)} 0x40051d <main>  
(gdb)
```

3. 掌握指针和类型转换相关的 C 编程

分析如下代码段（略），写出 gintr 和 intr 的结果，试着编译这段代码，如果遇到错误进行改正，并分析错误原因。

1. 在这个代码中，`gintr=((struct gatedesc *)&intr);`的意思是把 intr 的值进行强制类型转换，成为 struct gatedesc 类型，然后把这个值赋给 gintr。
`intr=((unsigned *)&gintr);`的意思是把 gintr 的后 32 位转换成 unsigned 类型，并赋给 intr。

gintr 的值为 ee0000010002, 而 intr 是 0x00010002unsigned 类型的。

2. 然后，我将这段代码使用 gcc 运行了这段代码，看到了 gcc 的报错

```
11.c: In function 'main':
11.c:45:8: warning: format '%llx' expects argument of type 'long long unsigned int', but argument 2 has type 'struct gatedesc' [-Wformat=]
printf("gintr is 0x%llx\n", gintr);
```

所以，我将 gintr 进行了强制类型转换。将 `printf("intr is 0x%llx\n", gintr);` 改成 `printf("intr is 0x%llx\n", *(long long unsigned *)&(gintr));`

```
gintr is 0xee0000010002
os@os-VirtualBox:~/下载$ gcc wj.c -o wj -I ./
os@os-VirtualBox:~/下载$ ./wj
intr is 0x10002
gintr is 0xee0000010002
os@os-VirtualBox:~/下载$
```

4. 掌握通用链表结构相关的 C 编程

查看 list.h 和 lab0_ex4.c，编写一个程序，利用 list.h 中的链表结构，将 26 个英文字母存入链表中，并逆序打印出来。

1. 我发现在 lab0_ex4.c 中引入了 list.h, 而在 list.t 中还引入了 defs.h, 所以我把 lab0_ex4.c, list.h, defs.h, 都放在了 home 下。

2. 我通过编译 lab0_ex4.c 这个文件得到了它的结果

```
os@os-VirtualBox:~$ gcc lab0_ex4.c -o lab0 -I ./
os@os-VirtualBox:~$ ./lab0_ex4
8
7
6
5
4
3
2
1
```

3. 将代码中某些部分修改，左图修改前，右图修改后。

```
struct entry {
    list_entry_t node;
    int num;
};

int main() {
    struct entry head;
    list_entry_t* p = &head.node;
    list_init(p);
    head.num = 0;
    int i;
    for (i = 1; i != 10; i++) {
        struct entry *e = (struct entry *)malloc(sizeof(struct entry));
        e->num = i;
        list_add(p, &(e->node));
        p = list_next(p);
    }
    //reverse list all node
    while ((p = list_prev(p)) != &head.node)
        printf("%d\n", ((struct entry *)p)->num);
    return 0;
}
```

```
struct entry {
    list_entry_t node;
    char character;
};

int main() {
    struct entry head;
    list_entry_t* p = &head.node;
    list_init(p);
    head.character = 'a';
    int i;
    for (i = 0; i <= 26; i++) {
        struct entry *e = (struct entry *)malloc(sizeof(struct entry));
        e->character = 'a'+i;
        list_add(p, &(e->node));
        p = list_next(p);
    }
    //reverse list all node
    while ((p = list_prev(p)) != &head.node)
        printf("%c ", ((struct entry *)p)->character);
    return 0;
}
```

4. 运行结果

```
os@os-VirtualBox:~$ gcc lab0_ex41.c -o lab01 -I ./
os@os-VirtualBox:~$ ./lab01
z y x w v u t s r q p o n m l k j i h g f e d c b a os@os-VirtualBox:~$
```