

第三次实验

181250090 刘育麟

第1题

在一个只有128M内存并且没有交换分区的计算机上，说说下面两个程序在编译及运行结果上的差别。

```
1 // 程序1
2 #define MEMSIZE 1024*1024
3 int count = 0;
4 void *p = NULL;
5 while(1) {
6     p = (void *)malloc(MEMSIZE);
7     if (!p) break;
8     printf("Current allocation %d MB\n", ++count);
9 }
10 // 程序2
11 #define MEMSIZE 1024*1024
12 int count = 0;
13 void *p = NULL;
14 while(1) {
15     p = (void *)malloc(MEMSIZE);
16     if (!p) break;
17     memset(p, 1, MEMSIZE);
18     printf("Current allocation %d MB\n", ++count);
19 }
```

第一道程序分配内存但没有填充，编译器可能会把内存分配优化掉，程序死循环；第二道，程序分配内存并进行填充，系统会一直分配内存，直到内存不足，退出循环

第2题

请使用程序设计语言(如C语言)，编程实现“ls”和“wc”命令，要求实现如下参数与功能(注: ls 的5个参数要求能组合，不需要实现参数连写，也不需要实现参数的任意次序; wc不需要实现参数，只要后面跟文件名即可)。ls -l(-d, -R, -a, -i)

wc [filename]

完成实现后，查找原命令的源代码，对其进行阅读分析，与自己的版本作比较，看有何不同，并将不同之处写入文档。

要求：

(1) 文档（包括specification文档和设计文档，可以二合一）言简意赅，简洁明了。

(2) 注意代码风格。

补充说明: 一些细节可以不实现，例如参数解析时-al的连写、ls -l在遇到链接文件时对链接文件的打印等等。要求实现同时输入多个参数，可以固定次序。

程序功能

- `ls -l(-d, -R, -a, -i)` 显示当前目录下的文件及其相关信息
 - i - inode 印出每个文件的 inode 号
 - l - 长 (long)。列举目录内容的细节，包括权限 (模式)、所有者、组群、大小、创建日期、文件是否是到系统其它地方的链接，以及链接的指向。
 - R - 递归 (recursive)。该选项递归地列举所有目录 (在当前目录之下) 的内容。
 - a - 全部 (all)。列举目录中的全部文件，包括隐藏文件 (.filename)。位于这个列表的起首处的 ... 和 . 依次是指父目录和你的当前目录。
 - d - directory 将目录像文件一样显示，而不是显示其下的文件。
- `wc [filename]` 统计文件的单词数、行数、所占字节数，多文件下，会分别输出文件统计数据 and 总和数据。

ls自己实现的关键代码

```
1 void output(struct dirent *dirent_pos, char *path) {
2     if (i_flag) {
3         printf("%20ld ", dirent_pos->d_ino);
4     }
5     if (l_flag) {
6         struct stat info;
7         char mode_str[11];
8         char *ctime();
9         if (stat(path, &info) == -1) {
10             perror(path);
11         } else {
12             mode_to_letters(info.st_mode, mode_str);
13             printf("%s ", mode_str);
14             printf("%4d ", (int) info.st_nlink);
15             printf("%-8s ", uid_to_name(info.st_uid));
16             printf("%-8s ", gid_to_name(info.st_gid));
17             printf("%8ld ", (long) info.st_size);
18             printf("%.14s ", 4 + ctime(&info.st_mtime));
19         }
20     }
21     printf("%s\n", dirent_pos->d_name);
22 }
23
24 void readDir(DIR *dir_ptr, char *path) {
25     struct dirent *dirent_pos;
26     int len = strlen(path);
27     DIR * tmp_pos[100];
28     char tmp_path[100][100];
29     int cnt = 0;
30     if (r_flag) printf("Source directory: %s\n", path);
31     while ((dirent_pos = readdir(dir_ptr)) != NULL) {
32
33         addPath(path, dirent_pos->d_name);
34         if (d_flag) {
35             output(dirent_pos, path);
36             return;
37         }
38         if (dirent_pos->d_name[0] == '.' && !a_flag) {
39             rmPath(path, len);
```

```

40         continue;
41     }
42
43     output(dirent_pos, path);
44     if (r_flag) {
45         if(strlen(dirent_pos->d_name) == 1 && dirent_pos->d_name[0] ==
'.') {
46             rmPath(path, len);
47             continue;
48         }
49         if(strlen(dirent_pos->d_name) == 2 && dirent_pos->d_name[0] ==
'.' && dirent_pos->d_name[1] == '.') {
50             rmPath(path, len);
51             continue;
52         }
53         // d_type : 8-文件, 4-目录
54         if (dirent_pos->d_type == 4) {
55             DIR *dir_ptr_next;
56             if ((dir_ptr_next = opendir(path)) != NULL) {
57                 memcpy(tmp_path[cnt], path, strlen(path));
58                 tmp_pos[cnt++] = dir_ptr_next;
59             }
60         }
61     }
62     rmPath(path, len);
63 }
64 if(r_flag){
65     for(int i = 0; i < cnt;++i){
66         readDir(tmp_pos[i], tmp_path[i]);
67     }
68 }
69 closedir(dir_ptr);
70 }

```

遍历DIR结构体，生成dirent结构体，将其解析为stat，可以获取该文件所有的属性。对文件属性进行处理之后就可以进行输出。并且每一层都需要保存该层的path，方便stat去寻找。

如果是-r则进行除了.和..所有文件夹递归搜索，如果是-a则所有.开头的文件都要输出。如果是-l则是在输出的时候要加上详细的输出，如果是-i则需要输出并加上inode号，如果是-d则只输出.文件。

wc自己实现的关键代码

```

1 void count(FILE *fp, char * fileName){
2     if(fp == NULL){
3         printf("wc: Cannot open %s.\n", fileName);
4         return;
5     }
6     int characters, lines, words, state;
7     char c;
8     state = characters = lines = words = 0;
9     while ((c = fgetc(fp)) != EOF) {
10         characters++;
11         if (c == '\n') {
12             lines++;
13             state = 0;
14             continue;
15         } else if (c == ' ') {

```

```

16         state = 0;
17         continue;
18     } else if (c == '\t') {
19         state = 0;
20         continue;
21     } else {
22         if (state == 0) {
23             state = 1;
24             words++;
25         }
26         continue;
27     }
28 }
29 printf("%d characters %d words %d lines %s\n", characters, words, lines,
30        fileName);

```

每一次读一个字符，并且对该字符进行判断，如果是\n的话则行数+1，如果是' '的话则state = 0，如果是\t的话与' '同理。而在state = 0时读到除了上面几个字符以外的其他字符的话，则state = 1并且单词数+1，以此类推可以获得字符数、单词数、行数。

与源码比较

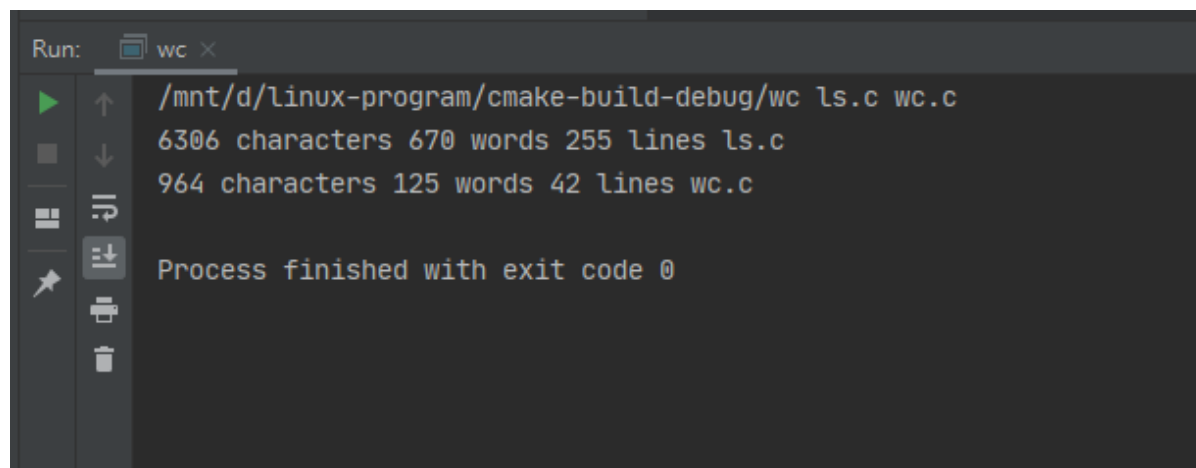
首先，源码进行了许多抽象，将很多操作交给了别的模块去做，几乎都是调用某个函数去执行，而调用的函数也与我写的代码有略微的不同，因为需要考虑越界的问题等边界问题，这些细微的东西都是我没有考虑到的。像是我们使用getopt，源码使用getopt_long。而使用的数据结构则几乎一致，都是使用DIR、dirent、stat去进行文件的解析与输出，所以可以说大体框架是相似的，但是源码更加完善与全面，可以预防编程时的错误情况避免程序中断或是系统崩溃。

其次，源码有更多的参数可以进行组合，而且也考虑了更多现实中不太会发生但是真实的程序却经常有的字符与可能性，像是wc里面就考虑了\f、\v、\r的可能，wc还使用goto语句，可以直接进行跳转，比我的程序看起来更加直接。我的wc逻辑是字符开始时就进行word++，而源码是遇到特殊字符后再进行判断。

实验结果

IDE使用CLion配置wsl进行代码的编写，控制台输出第一行是执行的应用与参数，第二行开始是输出。

WC



```

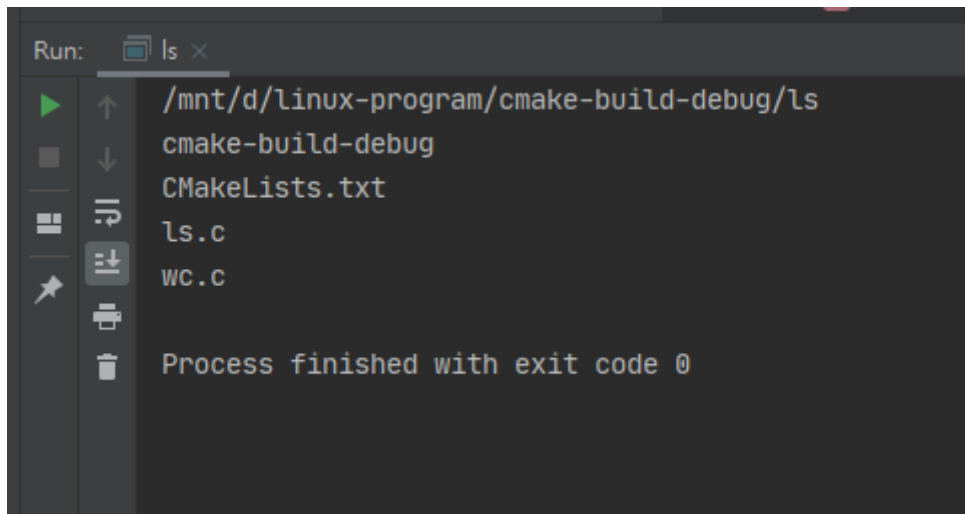
Run: wc x
/mnt/d/linux-program/cmake-build-debug/wc ls.c wc.c
6306 characters 670 words 255 lines ls.c
964 characters 125 words 42 lines wc.c
Process finished with exit code 0

```

ls

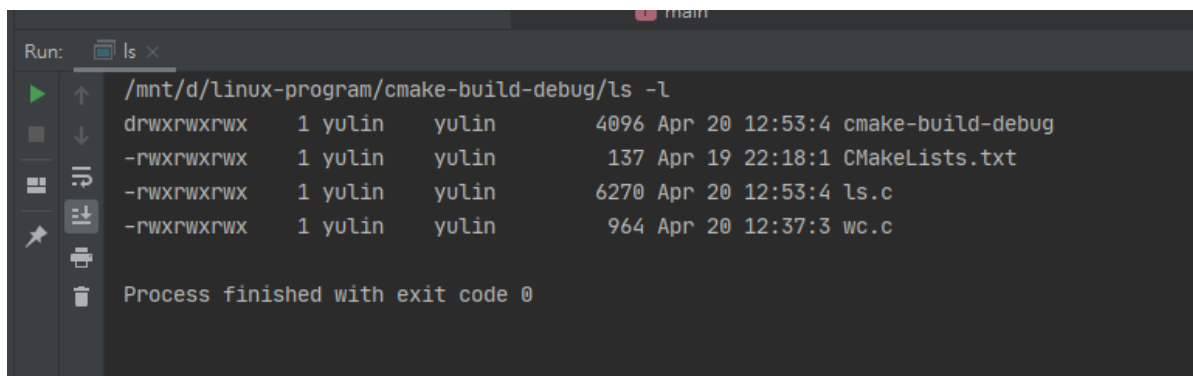
运行路径是linux-program

default



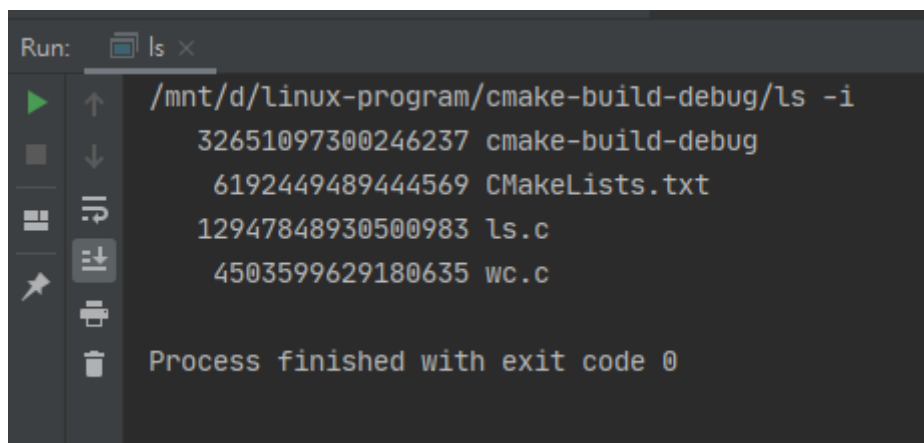
```
Run: ls x
/mnt/d/linux-program/cmake-build-debug/ls
cmake-build-debug
CMakeLists.txt
ls.c
wc.c
Process finished with exit code 0
```

-l



```
Run: ls x
/mnt/d/linux-program/cmake-build-debug/ls -l
drwxrwxrwx  1 yulin  yulin    4096 Apr 20 12:53:4 cmake-build-debug
-rwxrwxrwx  1 yulin  yulin    137 Apr 19 22:18:1 CMakeLists.txt
-rwxrwxrwx  1 yulin  yulin   6270 Apr 20 12:53:4 ls.c
-rwxrwxrwx  1 yulin  yulin    964 Apr 20 12:37:3 wc.c
Process finished with exit code 0
```

-i



```
Run: ls x
/mnt/d/linux-program/cmake-build-debug/ls -i
32651097300246237 cmake-build-debug
6192449489444569 CMakeLists.txt
12947848930500983 ls.c
4503599629180635 wc.c
Process finished with exit code 0
```

-R

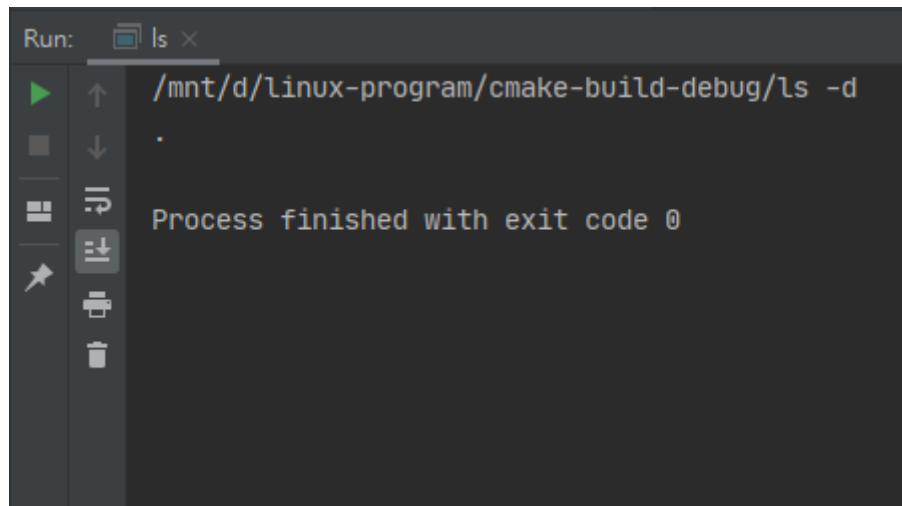
```
Run: ls x
/mnt/d/linux-program/cmake-build-debug/ls -R
Source directory: /mnt/d/linux-program
cmake-build-debug
CMakeLists.txt
ls.c
wc.c
Source directory: /mnt/d/linux-program/cmake-build-debug
CMakeCache.txt
CMakeFiles
cmake_install.cmake
linux_program
linux_program.cbp
ls
Makefile
wc
Source directory: /mnt/d/linux-program/cmake-build-debug/CMakeFiles
3.10.2
clion-environment.txt
clion-log.txt
cmake.check_cache
CMakeDirectoryInformation.cmake
CMakeOutput.log
CMakeTmp
feature_tests.bin
feature_tests.c
feature_tests.cxx
linux_program.dir
ls.dir
Makefile.cmake
Makefile2
progress.marks
TargetDirectories.txt
wc.dir
Source directory: /mnt/d/linux-program/cmake-build-debug/CMakeFiles/3.10.2
```

-a

```
Run: ls x
/mnt/d/linux-program/cmake-build-debug/ls -a
.
..
.idea
cmake-build-debug
CMakeLists.txt
ls.c
wc.c

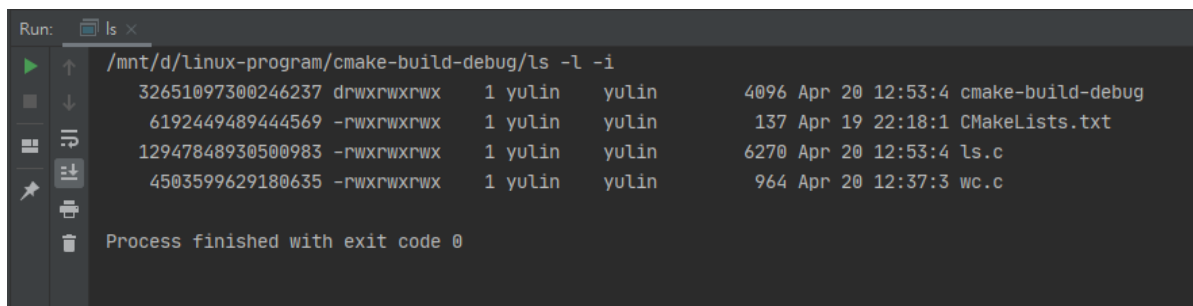
Process finished with exit code 0
```

-d



A terminal window titled 'Run: ls' showing the command `/mnt/d/linux-program/cmake-build-debug/ls -d` and its output `.`. The terminal also displays 'Process finished with exit code 0'.

-l-i



A terminal window titled 'Run: ls' showing the command `/mnt/d/linux-program/cmake-build-debug/ls -l -i` and its output. The output lists four files with their permissions, owner, group, size, date, and filename.

Permissions	Owner	Group	Size	Date	Filename
drwxrwxrwx	yulin	yulin	4096	Apr 20 12:53:4	cmake-build-debug
-rwxrwxrwx	yulin	yulin	137	Apr 19 22:18:1	CMakeLists.txt
-rwxrwxrwx	yulin	yulin	6270	Apr 20 12:53:4	ls.c
-rwxrwxrwx	yulin	yulin	964	Apr 20 12:37:3	wc.c

The terminal also displays 'Process finished with exit code 0'.

-R -l-i -a

```
Run: ls x
/mnt/d/linux-program/cmake-build-debug/ls -R -l -i -a
Source directory: /mnt/d/linux-program
 6755399442865875 drwxrwxrwx 1 yulin yulin 4096 Apr 20 12:53:4 .
 6755399442865875 drwxrwxrwx 1 yulin yulin 4096 Apr 19 22:22:2 ..
 5629499536023254 drwxrwxrwx 1 yulin yulin 4096 Apr 20 13:02:0 .idea
 32651097300246237 drwxrwxrwx 1 yulin yulin 4096 Apr 20 12:53:4 cmake-build-debug
 6192449489444569 -rwxrwxrwx 1 yulin yulin 137 Apr 19 22:18:1 CMakeLists.txt
 12947848930500983 -rwxrwxrwx 1 yulin yulin 6270 Apr 20 12:53:4 ls.c
 4503599629180635 -rwxrwxrwx 1 yulin yulin 964 Apr 20 12:37:3 wc.c
Source directory: /mnt/d/linux-program/.idea
 5629499536023254 drwxrwxrwx 1 yulin yulin 4096 Apr 20 13:02:0 .
 5629499536023254 drwxrwxrwx 1 yulin yulin 4096 Apr 20 12:53:4 ..
 1970324838785406 -rwxrwxrwx 1 yulin yulin 215 Apr 19 12:10:1 .gitignore
 1407374885364099 -rwxrwxrwx 1 yulin yulin 13 Apr 19 12:10:3 .name
 3377699722338646 -rwxrwxrwx 1 yulin yulin 442 Apr 19 12:10:1 deployment.xml
 2251799815496059 -rwxrwxrwx 1 yulin yulin 98 Apr 19 12:10:3 linux-program.iml
 2533274792206716 -rwxrwxrwx 1 yulin yulin 140 Apr 19 12:10:1 misc.xml
 3096224745627989 -rwxrwxrwx 1 yulin yulin 285 Apr 19 12:10:1 modules.xml
 4222124652469976 -rwxrwxrwx 1 yulin yulin 10270 Apr 20 13:02:0 workspace.xml
Source directory: /mnt/d/linux-program/cmake-build-debug
 32651097300246237 drwxrwxrwx 1 yulin yulin 4096 Apr 20 12:53:4 .
 32651097300246237 drwxrwxrwx 1 yulin yulin 4096 Apr 20 12:53:4 ..
 3940649675760013 -rwxrwxrwx 1 yulin yulin 34670 Apr 19 14:05:5 CMakeCache.txt
 1688849862074083 drwxrwxrwx 1 yulin yulin 4096 Apr 20 13:02:0 CMakeFiles
 3096224745628019 -rwxrwxrwx 1 yulin yulin 1509 Apr 19 12:10:1 cmake_install.cmake
 23362423068795271 -rwxrwxrwx 1 yulin yulin 19144 Apr 19 22:16:0 linux_program
 22517998138663267 -rwxrwxrwx 1 yulin yulin 7837 Apr 19 22:18:1 linux_program.cbp
 5629499536023980 -rwxrwxrwx 1 yulin yulin 19400 Apr 20 12:53:4 ls
 3096224745628061 -rwxrwxrwx 1 yulin yulin 5502 Apr 19 22:18:1 Makefile
 1688849862074792 -rwxrwxrwx 1 yulin yulin 11368 Apr 20 12:29:2 wc
Source directory: /mnt/d/linux-program/cmake-build-debug/CMakeFiles
 1688849862074083 drwxrwxrwx 1 yulin yulin 4096 Apr 20 13:02:0 .
 1688849862074083 drwxrwxrwx 1 yulin yulin 4096 Apr 20 12:53:4 ..
 1970324838784859 drwxrwxrwx 1 yulin yulin 4096 Apr 19 14:05:5 3.10.2
 1970324838784758 -rwxrwxrwx 1 yulin yulin 117 Apr 19 22:18:1 clion-environment.txt
 2251799815496061 -rwxrwxrwx 1 yulin yulin 209 Apr 19 22:18:1 clion-log.txt
 3096224745627995 -rwxrwxrwx 1 yulin yulin 85 Apr 19 22:18:1 cmake.check_cache
 3096224745628017 -rwxrwxrwx 1 yulin yulin 634 Apr 19 12:10:1 CMakeDirectoryInformation.cmake
 3096224745627389 -rwxrwxrwx 1 yulin yulin 44807 Apr 19 14:05:5 CMakeOutput.log
 4785074605891922 drwxrwxrwx 1 yulin yulin 4096 Apr 19 14:05:5 CMakeTmp
```

第3题

请用中文描述一下字符型Linux驱动模块的编写以及编译过程？

驱动函数没有main函数。模块在调用insmod命令时被加载。入口点是init_module函数，通常在该函数中完成设备的注册，而模块调用rmmod函数时被卸载，此时的入口点是cleanup_module函数，该函数中完成设备的卸载。在设备完成注册加载后，用户就可以对设备进行操作，驱动程序就是用于实现这些操作，在用户应用程序调用相应入口函数时执行相关的操作，init_module入口函数点则不需要完成其他如read、write之类的功能。

设备驱动程序的入口点还需要被定义在<linux/fs.h>的struct_file结构中。

字符型设备驱动程序的几个主要组成：

(1) 设备注册

设备注册使用函数register_chrdev，调用该函数后就可以向系统申请主设备号，如果register_chrdev操作成功，设备名就会出现在/proc/devices文件里。

(2) 设备解除注册

在关闭设备时，通常需要解除原先的设备注册，此时可以用函数unregister_chrdev，此后该设备就会从/proc/devices消失。

(3) 打开设备

打开设备的接口函数是open，根据设备不同，open函数完成的功能主要是下面几个：递增计数器、检查特定设备的特殊情况、初始化设备、识别次设备号。

(4) 释放设备

释放设备的接口函数是`release`。就是该进程展示停止对该设备的使用。要完成的工作有：递减计数器、在最后一次释放设备操作时关闭设备。

(5) 读写设备

主要任务就是将内核空间的数据复制到用户空间，或者从用户空间复制到内核空间，也就是将内核空间缓冲区里的数据复制到用户空间的缓冲区中或者相反。

(6) 获取内存

设备驱动程序开辟内存分为基于内存地址和基于页面为单位。基于内存地址的函数是`kmalloc`，返回的时物理地址，而`malloc`等返回的时线性地址，因此在驱动程序中不能使用`malloc`函数。与`malloc()`不同，`kmalloc()`申请的空间大小有限制，长度是2的整次方，并且不会对所获取的内存空间清零。

(7) 打印信息

就如同在编写用户空间的应用程序，打印信息又是是很好的调式手段，也是在代码中很常用的组成部分。但是与用户空间不同，在内核空间中要用函数`printk`为不能用平常的函数`printf`。`printk`和`printf`很类似，都可以按照一定的格式打印消息，所不同的是，`printk`还可以定义打印消息的优先级。

proc文件系统

`/proc`文件系统是一个伪文件系统，它是一种内核和内核模块用来向进程发送信息的机制。这个伪文件系统让用户可以和内核内部数据结构进行交互，获取有关进程的有用信息，在运行时通过改变内核参数改变设置。与其他文件系统不同，`/proc`存在于内存中而不是硬盘上，读者可以查看`/proc`文件系统的内容。`/proc`文件系统体现了内核及进场运行的内容，在加载模块成功后，读者可以使用查看`/proc/devices`文件获得相关设备的主设备号。

驱动模块的编译

驱动程序实际属于内核的一部分，那么在编译的时候就需要使用已经编译好的内核，来编译驱动程序了，这点尤为重要。

像内核编译过程一样，可以将内核模块编译为`module`的方式编译，在运行时加载该模块即可，而不用每次都需要完整的对内核进行编译。驱动模块大部分使用这种方法。