第三次实验

191250133 陶泽华

第一题

memset的作用是将某一块内存中的内容全部设置为指定的值， 这个函数通常为新申请的内存做初始化工作。

程序1: 没有使用memset，分配内存但没有将内容设置为指定的值，编译器可能会把内存分配优化掉，程序死循环

程序2: 使用memset后程序分配内存并进行赋值，内存会一直被占用，直到系统内存不足，退出循环

第二题

ls指令

实现代码如下

```
int l_flag = 0, R_flag = 0, d_flag = 0, a_flag = 0, i_flag = 0;

void mode_to(int mode, char str[]) {
    strcpy(str, "----------");
    if (S_ISDIR(mode)) str[0] = 'd';
    if (S_ISCHR(mode)) str[0] = 'c';
    if (S_ISBLK(mode)) str[0] = 'b';
    if ((mode & S_IRUSR)) str[1] = 'r';
    if ((mode & S_IWUSR)) str[2] = 'w';
    if ((mode & S_IXUSR)) str[3] = 'x';
    if ((mode & S_IRGRP)) str[4] = 'r';
    if ((mode & S_IWGRP)) str[5] = 'w';
    if ((mode & S_IXGRP)) str[6] = 'x';
    if ((mode & S_IROTH)) str[7] = 'r';
    if ((mode & S_IWOTH)) str[8] = 'w';
    if ((mode & S_IXOTH)) str[9] = 'x';
}

// 递归得到所有目录的路径
void recursion(char *path) {
    DIR *dir;
    struct dirent *ptr;
    dir = opendir(path);
    while ((ptr = readdir(dir)) != NULL) {
        if (strcmp(ptr->d_name, ".") != 0 && strcmp(ptr->d_name, "..") != 0) {
            // 8 是文件，4 是目录
            if (ptr->d_type == 4) {
                char tmpPath[1000];
                strcpy(tmpPath, path);
                strcat(tmpPath, "/");
```

```
                strcat(tmpPath, ptr->d_name);
                DIR *tmpDir;
                tmpDir = opendir(tmpPath);
                printf("%s:\n", tmpPath);
                struct dirent *ptr2;
                while ((ptr2 = readdir(tmpDir)) != NULL) {
                    char tmp[1000];
                    strcpy(tmp,tmpPath);
                    strcat(tmp, "/");
                    strcat(tmp, ptr2->d_name);
                    struct stat buf3;
                    stat(tmp, &buf3);
                    struct passwd *pwd2;
                    pwd2 = getpwuid(buf3.st_uid);
                    struct group *grp2;
                    grp2 = getgrgid(buf3.st_gid);
                    char str[11];
                    mode_to(buf3.st_mode, str);
                    if (a_flag != 1) {
                        if (strcmp(ptr2->d_name, ".") != 0 && strcmp(ptr2->d_name,
"..") != 0) {
                            if (i_flag == 1) {
                                printf("%llu ", ptr2->d_ino);
                            }
                            if (l_flag == 1) {
                                printf("%s %d %s %s %lld %.12s ", str, buf3.st_nlink,
                                       pwd2->pw_name, grp2->gr_name, buf3.st_size, 4 +
ctime(&buf3.st_mtimespec));
                            }
                            printf("%s\n", ptr2->d_name);
                        }
                    } else if (a_flag == 1) {
                        if (i_flag == 1) {
                            printf("%llu ", ptr2->d_ino);
                        }
                        if (l_flag == 1) {
                            printf("%s %d %s %s %lld %.12s ", str, buf3.st_nlink,
                                   pwd2->pw_name, grp2->gr_name, buf3.st_size, 4 +
ctime(&buf3.st_mtimespec));
                        }
                        printf("%s\n", ptr2->d_name);
                    }
                }
                printf("\n\n");
                closedir(tmpDir);
                recursion(tmpPath);
            }
        }
    }
```

```
    closedir(dir);
}

void ls() {
    DIR *dir;
    char *path = getenv("PWD");
    struct dirent *ptr;
    dir = opendir(path);
    while ((ptr = readdir(dir)) != NULL) {
        char tmpPath[1000];
        strcpy(tmpPath, path);
        strcat(tmpPath, "/");
        strcat(tmpPath, ptr->d_name);
        struct stat buf;
        stat(tmpPath, &buf);
        struct passwd *pwd;
        pwd = getpwuid(buf.st_uid);
        struct group *grp;
        grp = getgrgid(buf.st_gid);
        char str[11];
        mode_to(buf.st_mode, str);
        if (d_flag == 1 && strcmp(ptr->d_name, ".") == 0) {
            if (i_flag == 1) {
                printf("%llu ", ptr->d_ino);
            }
            if (l_flag == 1) {
                printf("%s %d %s %s %lld %.12s ", str, buf.st_nlink,
                       pwd->pw_name, grp->gr_name, buf.st_size, 4 +
ctime(&buf.st_mtimespec));
            }
            printf("%s\n", ptr->d_name);
            break;
        } else {
            if (a_flag != 1) {
                if (strcmp(ptr->d_name, ".") != 0 && strcmp(ptr->d_name, "..") != 0) {
                    if (i_flag == 1) {
                        printf("%llu ", ptr->d_ino);
                    }
                    if (l_flag == 1) {
                        printf("%s %d %s %s %lld %.12s ", str, buf.st_nlink,
                               pwd->pw_name, grp->gr_name, buf.st_size, 4 +
ctime(&buf.st_mtimespec));
                    }
                    printf("%s\n", ptr->d_name);
                }
            } else if (a_flag == 1) {
                if (i_flag == 1) {
                    printf("%llu ", ptr->d_ino);
                }
```

```
                if (l_flag == 1) {
                    printf("%s %d %s %s %lld %.12s ", str, buf.st_nlink,
                           pwd->pw_name, grp->gr_name, buf.st_size, 4 +
ctime(&buf.st_mtimespec));
                }
                printf("%s\n", ptr->d_name);
            }
        }
    }
    if (R_flag == 1 && d_flag != 1) {
        printf("\n\n");
        recursion(path);
    }
    closedir(dir);
}


int main() {
    char command[100];
    gets(command);
    for (int j = 2; j <= strlen(command); j++) {
        if (command[j] == 'l') l_flag = 1;
        else if (command[j] == 'd') d_flag = 1;
        else if (command[j] == 'a') a_flag = 1;
        else if (command[j] == 'R') R_flag = 1;
        else if (command[j] == 'i') i_flag = 1;
    }
    ls();
    return 0;
}
```

首先从命令行读取指令并根据输入的指令将对应的标志位设置为1，ls方法会根据不同的标志位将需要的输出进行组合。

测试：（运行路径是`/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug`）

ls

```
/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/code
warning: this program uses gets(), which is unsafe.
ls
CMakeFiles
Makefile
cmake_install.cmake
code
CMakeCache.txt
code.cbp
```

ls -l

```
/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/code
warning: this program uses gets(), which is unsafe.
ls -l
drwxr-xr-x 15 taozehua staff 480 Mar 30 10:36 CMakeFiles
-rw-r--r-- 1 taozehua staff 5119 Mar 29 12:44 Makefile
-rw-r--r-- 1 taozehua staff 1468 Mar 29 12:44 cmake_install.cmake
-rwxr-xr-x 1 taozehua staff 51464 Mar 30 00:05 code
-rw-r--r-- 1 taozehua staff 22578 Mar 29 12:44 CMakeCache.txt
-rw-r--r-- 1 taozehua staff 6322 Mar 29 12:44 code.cbp
```

ls -i

```
/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/code
warning: this program uses gets(), which is unsafe.
ls -i
63889952 CMakeFiles
63890029 Makefile
63890031 cmake_install.cmake
64053702 code
63890020 CMakeCache.txt
63890033 code.cbp
```

ls -R

code

```
/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/code
warning: this program uses gets(), which is unsafe.
ls -R
CMakeFiles
Makefile
cmake_install.cmake
code
CMakeCache.txt
code.cbp


/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/CMakeFiles:
cmake.check_cache
3.15.3
CMakeOutput.log
CMakeError.log
Makefile.cmake
code.dir
CMakeTmp
progress.marks
TargetDirectories.txt
CMakeDirectoryInformation.cmake
clion-log.txt
clion-environment.txt
Makefile2


/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/CMakeFiles/3.15.3:
CMakeDetermineCompilerABI_C.bin
CompilerIdC
CMakeCCompiler.cmake
CMakeSystem.cmake


/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/CMakeFiles/3.15.3/CompilerIdC:
```

ls -a

code

```
/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/code
warning: this program uses gets(), which is unsafe.
ls -a
.
..
CMakeFiles
Makefile
cmake_install.cmake
code
CMakeCache.txt
code.cbp
```

ls -d

code

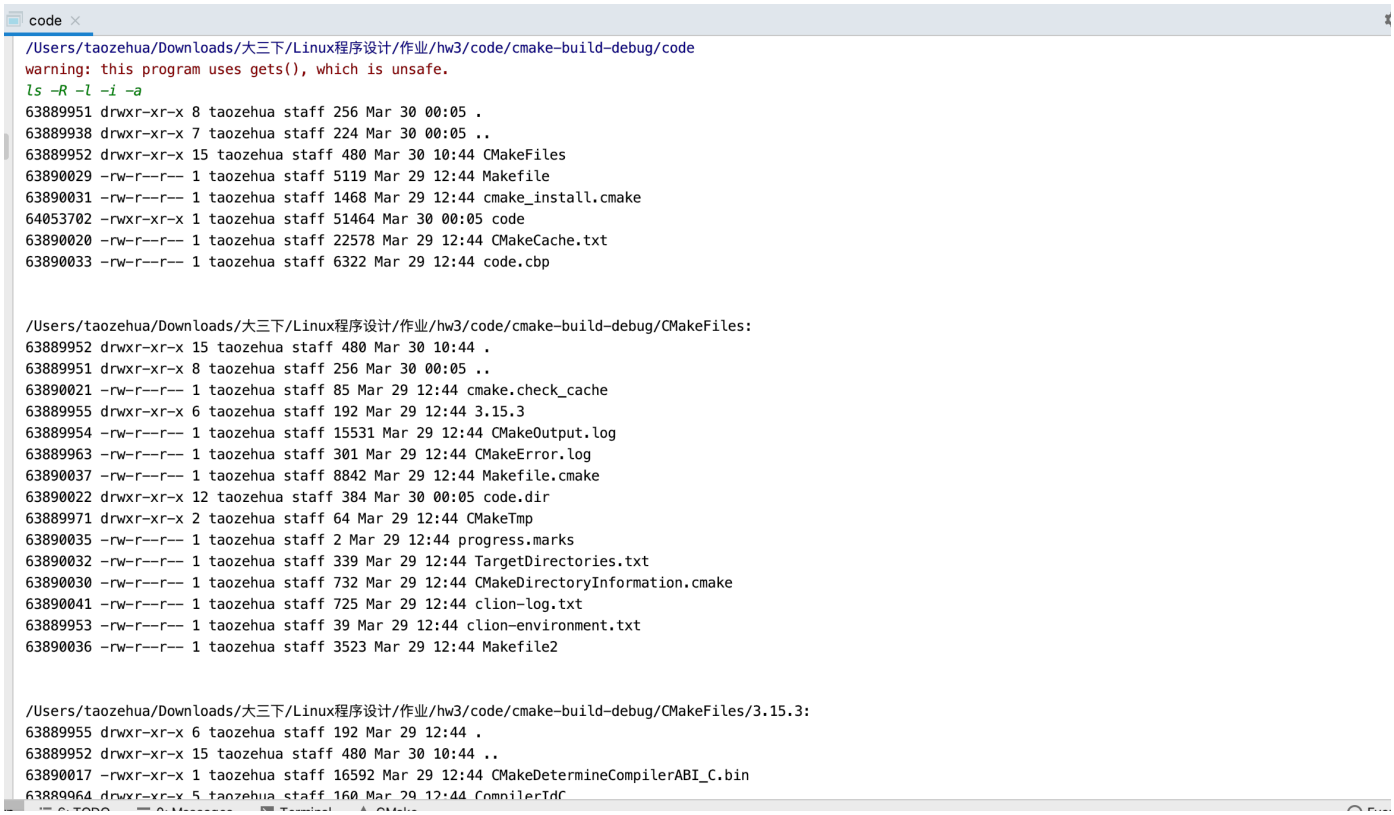
```
/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/code
warning: this program uses gets(), which is unsafe.
ls -d
.

Process finished with exit code 0
```

ls -l -i

code

```
/Users/taozehua/Downloads/大三下/Linux程序设计/作业/hw3/code/cmake-build-debug/code
warning: this program uses gets(), which is unsafe.
ls -l -i
63889952 drwxr-xr-x 15 taozehua staff 480 Mar 30 10:43 CMakeFiles
63890029 -rw-r--r-- 1 taozehua staff 5119 Mar 29 12:44 Makefile
63890031 -rw-r--r-- 1 taozehua staff 1468 Mar 29 12:44 cmake_install.cmake
64053702 -rwxr-xr-x 1 taozehua staff 51464 Mar 30 00:05 code
63890020 -rw-r--r-- 1 taozehua staff 22578 Mar 29 12:44 CMakeCache.txt
63890033 -rw-r--r-- 1 taozehua staff 6322 Mar 29 12:44 code.cbp
```

ls -R -l -i -a



ls -R -l -i -d

wc指令

实现如下，该方法需要传入文件路径和文件名称，逐一读取字符并对需要的结果进行统计

```
void wordCount(char *path, char * filename){
    FILE *file = fopen(path, "r");
    char c;
    int bytes=0,lines=0,words=0;
    int whetherInWord=0; // 0不在单词中，1在单词中
    while((c = fgetc(file)) != EOF){
        bytes++;
        if(c=='\n'){
            lines++;
            whetherInWord=0;
        }else if(c==' ' ||  c=='\t'){
            whetherInWord=0;
        }else if(c!=' '){
            if(whetherInWord==0){
                whetherInWord=1;
                words++;
```

```
            }
        }
    }
    fclose(file);
    printf("%d %d %d %s\n", lines,words,bytes,filename);
}
```

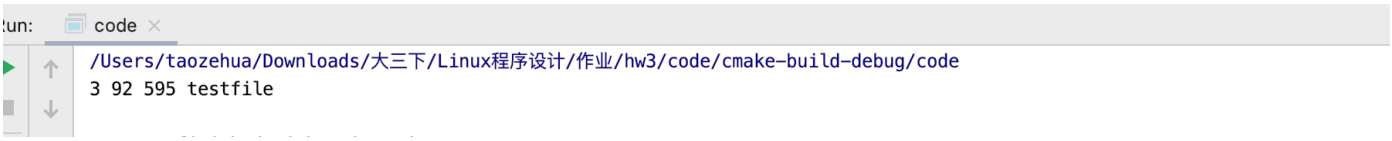
测试:

使用如下文件进行测试：

```
Linux networks are becoming more and more common, but scurity is often an overlooked
issue. Unfortunately, in today's environment all networks are potential hacker targets,
fro0m tp-secret military research networks to small home LANs.
Linux Network Securty focuses on securing Linux in a networked environment, where the
security of the entire network needs to be considered rather than just isolated
machines.
It uses a mix of theory and practicl techniques to teach administrators how to install
and use security applications, as well as how the applcations work and why they are
necesary.
```

使用wc命令输出如下：

程序输出如下：



与源码比较

源码实现时进行了许多抽象，将许多操作封装成函数抽象出去，在代码中进行调用。而在我的代码中一些可以复用的操作没有被抽象出去，造成了代码出现重复。对于ls指令源码可以将更多的参数进行组合，实现得更加灵活，组合起来更加方便。另外源码考虑了边界问题，更加完善与全面，避免运行时造成程序崩溃。

第三题

驱动程序没有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，在设备完成注册加载后，用户就可以对设备进行操作。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。

编写过程：

1. 指定版本信息
2. 定义数据结构`struct file_operations`，将系统调用和驱动程序关联起来（file_operations结构的每一个成员的名字都对应着一个系统调用）
3. 编写驱动程序，填充file_operations的各个域
4. 注册设备驱动程序，使用register_chrdev注册字符型设备

5. 在/dev文件系统中用mknod创建设备文件， 并将该文件绑定到设备号上

编译过程：

驱动程序实际属于内核的一部分，在编译的时候需要使用已经编译好的内核来编译驱动程序。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。