

Lab 2. Linux C Program

1. Mục đích

Thông qua bài thực hành này, sinh viên sẽ nắm vững các kiến thức sau:

- Cách xử lý argument truyền vào chương trình C
- Build và run chương trình C trên Linux
- Sử dụng các system call cơ bản của Linux để làm việc với file và process

2. Tóm tắt lý thuyết

2.1. Chương trình C

Chương trình C bắt đầu thực thi từ hàm main().

Prototype đầy đủ của hàm main() có dạng *int main (int argc, char* argv[])*, trong đó:

- argc là số lượng tham số, cũng chính là kích thước của mảng argv[]
- argv[] là mảng của các con trỏ trỏ tới các tham số theo định dạng chuỗi kí tự
- Theo quy ước argv[0] là tham số chứa tên command
- Theo quy ước hàm main() return 0 nếu không có lỗi xảy ra và return != 0 nếu có lỗi (thường là mã lỗi có giá trị từ 1 => 255)

gcc (GNU Compiler Collection) là compiler phổ biến trên các hệ điều hành Linux. Các file source code C được compile bằng gcc theo command:

```
gcc <file1.c> <file2.c> <...> -I <header dir> -o <output file name>
```

Nếu <output file name> không được chỉ định, a.out sẽ được dùng như tên mặc định

Kết quả sau khi compile là chương trình có thể thực thi được. Thực thi chương trình này bằng cách nhập đường dẫn của nó vào Terminal và nhấn Enter. Ví dụ như: `./<output file name>`

2.2. Các system call và library function cơ bản

2.2.1. Làm việc với file/dir

```
#include <fcntl.h>

int open(const char *path, int oflag, ... /* mode_t mode */ );

int openat(int fd, const char *path, int oflag, ... /* mode_t mode */ );

Both return: file descriptor if OK, -1 on error
```

```
#include <unistd.h>

int close(int fd);

Returns: 0 if OK, -1 on error
```

```
#include <unistd.h>
```

```
off_t lseek(int fd, off_t offset, int whence);
```

Returns: new file offset if OK, -1 on error

```
#include <unistd.h>
```

```
ssize_t read(int fd, void *buf, size_t nbytes);
```

Returns: number of bytes read, 0 if end of file, -1 on error

```
#include <unistd.h>
```

```
ssize_t write(int fd, const void *buf, size_t nbytes);
```

Returns: number of bytes written if OK, -1 on error

```
#include <unistd.h>
```

```
int dup(int fd);
```

return: new file descriptor if OK, -1 on error

```
#include <sys/stat.h>
```

```
int stat(const char *restrict pathname, struct stat *restrict buf);  
return: 0 if OK, -1 on error
```

```
#include <dirent.h>
```

```
DIR *opendir(const char *pathname);
```

return: pointer if OK, NULL on error

```
struct dirent *readdir(DIR *dp);
```

Returns: pointer if OK, NULL at end of directory or error

```
int closedir(DIR *dp);
```

Returns: 0 if OK, -1 on error

2.2.2. Làm việc với process

```
#include <unistd.h>
```

```
pid_t fork(void);
```

Returns: 0 in child, process ID of child in parent, -1 on error

```
#include <sys/wait.h>
```

```
pid_t wait(int *statloc);
```

Both return: process ID if OK, 0 (see later), or -1 on error

```
#include <unistd.h>

int pipe(int fd[2]);
```

Returns: 0 if OK, -1 on error

```
#include <unistd.h>
int execv(const char *pathname, char *const argv[]);
return: -1 on error, no return on success
```

3. Thực hành

Tạo thư mục làm việc lab2 trong /home/fetel

3.1. Chương trình mycat

Chương trình mycat có chức năng tương tự như command cat, nó đọc các file từ đường dẫn là các tham số truyền vào hoặc đọc từ stdin nếu không nhận được đường dẫn. Nội dung của file đã đọc được in ra stdout. Lỗi (nếu có) được in ra stderr.

Tạo file mycat.c trong thư mục lab2 có nội dung như bên dưới. Compile và thực thi chương trình này.

Ví dụ: ./mycat ./mycat.c

```
1  #include <unistd.h>
2  #include <fcntl.h>
3  #include <stdio.h>
4  #include <errno.h>
5
6  #define FD_STDIN      0
7  #define FD_STDOUT     1
8  #define FD_STDERR     2
9
10 /* System calls used in this program:
11     1) open: open file
12     2) close: close file
13     3) write: write to file descriptor
14     4) read: read from file descriptor
15     5) _exit: terminate program
16 Library functions used in this program:
17     1) perror: print error message
18 */
19
20 void mycat(int fd){
21     while(1){
22         char c;
23         if(read(fd, &c, 1) == 1){
24             if(write(FD_STDOUT, &c, 1) != 1){
```

```

25         perror("Error");
26         _exit(errno);
27     }
28     } else {
29         break;
30     }
31 }
32 }
33
34 int main(int argc, char* argv[]){
35     if(argc == 1){ //read file from stdin
36         mycat(FD_STDIN);
37     } else { //read files from pathnames
38         for(int i = 1; i < argc; i++){
39             int fd = open(argv[i], O_RDONLY);
40             if(fd > 0) {
41                 mycat(fd);
42                 close(fd);
43             } else {
44                 perror("Error");
45                 _exit(errno);
46             }
47         }
48     }
49     return 0;
50 }

```

3.2. Chương trình myls

Chương trình myls có chức năng tương tự như command ls -al, nó liệt kê thông tin chi tiết của file/dir có đường dẫn là tham số truyền vào. Lỗi nếu có được in ra stderr.

Tạo file myls.c trong thư mục lab2 có nội dung như bên dưới. Compile và thực thi chương trình này. So sánh kết quả thu được với command ls -al

Ví dụ:

```
ls -al ~
```

```
./mysls ~
```

```
ls -al myls
```

```
./mysls myls
```

```

1  #include <unistd.h>
2  #include <dirent.h>
3  #include <sys/stat.h>

```

```
4  #include <stdio.h>
5  #include <pwd.h>
6  #include <grp.h>
7  #include <time.h>
8  #include <errno.h>
9
10 #define FD_STDIN      0
11 #define FD_STDOUT     1
12 #define FD_STDERR     2
13
14 #define FILE_INFO_MAX_LEN  1000
15 #define PATH_NAME_MAX_LEN  1000
16
17 /* System calls used in this program:
18     1) write: write to file descriptor
19     2) stat: get file status
20     3) _exit: terminate program
21 Library functions used in this program:
22     1) perror: print error message
23     2) sprintf: construct string in specified format
24     3) getpwuid: get password file entry
25     4) getgrgid: get group file entry
26     5) ctime: construct date & time string
27     6) opendir: open dir
28     7) closedir: close dir
29     8) readdir: read dir entry
30 */
31
32 void print(char* str){
33     while(*str){
34         if(write(FD_STDOUT, str++, 1) != 1){
35             perror("Error");
36             _exit(errno);
37         }
38     }
39 }
40
41 void filestat(char* pathname, struct stat* buf){
42     if(stat(pathname, buf) != 0){
43         perror("Error");
44         _exit(errno);
45     }
46 }
47
48 void myls(char* filename, struct stat* buf){
```

```
49 char info[FILE_INFO_MAX_LEN];
50 int len = 0;
51
52 // type
53 if(S_ISREG(buf->st_mode)){
54     info[len++] = '-';
55 } else if(S_ISDIR(buf->st_mode)){
56     info[len++] = 'd';
57 } else if(S_ISCHR(buf->st_mode)){
58     info[len++] = 'c';
59 } else if(S_ISBLK(buf->st_mode)){
60     info[len++] = 'b';
61 } else if(S_ISFIFO(buf->st_mode)){
62     info[len++] = 'p';
63 } else if(S_ISLNK(buf->st_mode)){
64     info[len++] = 'l';
65 } else { // socket
66     info[len++] = 's';
67 }
68
69 // permissions
70 char modes[] = {'r', 'w', 'x'};
71 int mode_mask[] = {
72     S_IRUSR, S_IWUSR, S_IXUSR,
73     S_IRGRP, S_IWGRP, S_IXGRP,
74     S_IROTH, S_IWOTH, S_IXOTH
75 };
76 for(int i = 0; i < 3; i++){
77     for(int j = 0; j < 3; j++){
78         if(buf->st_mode & mode_mask[3*i + j]){
79             info[len++] = modes[j];
80         } else {
81             info[len++] = '-';
82         }
83     }
84 }
85
86 len += sprintf(info + len, " %3ld", buf->st_nlink); // number of hard links
87 len += sprintf(info + len, " %7s", getpwuid(buf->st_uid)->pw_name); // owner
88 len += sprintf(info + len, " %7s", getgrgid(buf->st_gid)->gr_name); // group
89 len += sprintf(info + len, " %7ld", buf->st_size); // size
90 len += sprintf(info + len, " %s", ctime(&buf->st_mtime)); // last modified date
91 len--; // remove new line char
92 len += sprintf(info + len, " %s", filename); // filename
93
```

```

94     info[len++] = '\n';
95     info[len] = 0;
96     print(info);
97 }
98
99 int main(int argc, char* argv){
100
101     char* target;
102     char wd[] = ".";
103     char pathname[PATH_NAME_MAX_LEN];
104     struct stat buf;
105
106     target = (argc == 1) ? wd : argv[1];
107     filestat(target, &buf);
108
109     if(S_ISDIR(buf.st_mode)){// list dir
110         DIR* dir = opendir(target);
111         struct dirent* dirent = readdir(dir);
112         while(dirent != 0){
113             sprintf(pathname, "%s/%s", target, dirent->d_name);
114             filestat(pathname, &buf);
115             myls(dirent->d_name, &buf);
116             dirent = readdir(dir);
117         }
118         closedir(dir);
119     } else { // list file
120         myls(target, &buf);
121     }
122
123     return 0;
124 }

```

3.3. Chương trình myfork

Chương trình myfork tạo ra các process con là bản sao của chính nó để thực thi các chương trình khác. Các process con có stdout được nối với pipes của process cha. Process cha sẽ đọc các pipes và in kết quả của từng process con theo thứ tự và định dạng nhất định. Lỗi nếu có được in ra stderr.

Tạo file myfork.c trong thư mục lab2 có nội dung như bên dưới. Compile và thực thi chương trình này.

Ví dụ: `./myfork`

```

1  #include <unistd.h>
2  #include <stdio.h>
3  #include <errno.h>
4  #include <sys/wait.h>

```

```
5
6 #define FD_STDIN      0
7 #define FD_STDOUT     1
8 #define FD_STDERR     2
9
10 #define CMD_MAX_LEN   10
11
12 /* System calls used in this program:
13     1) write: write to file descriptor
14     2) read: read from file descriptor
15     3) pipe: create pipe
16     4) fork: create child process
17     5) close: close a file descriptor
18     6) dup: duplicate file descriptor
19     7) _exit: terminate program
20     8) wait: wait for child process to terminate
21 Library functions used in this program:
22     1) perror: print error message
23     2) execv: execute a file
24 */
25
26 void print(char* str){
27     while(*str){
28         if(write(FD_STDOUT, str++, 1) != 1){
29             perror("Error");
30             _exit(errno);
31         }
32     }
33 }
34
35 void printfd(int fd){
36     while(1){
37         char c;
38         if(read(fd, &c, 1) == 1){
39             if(write(FD_STDOUT, &c, 1) != 1){
40                 perror("Error");
41                 _exit(errno);
42             }
43         } else {
44             break;
45         }
46     }
47 }
48
49 void myfork(char* argv[], int fd[2]){
```



```
50     pipe(fd);
51     if(fork() == 0){ //child process
52         close(FD_STDOUT);
53         dup(fd[1]);
54         execv(argv[0], argv);
55         _exit(errno); // should not get here
56     }
57 }
58
59 int main(int argc, char* argv[]){
60     char* cmd[][CMD_MAX_LEN] = {
61         {"/usr/bin/uname", "-a", 0},
62         {"/usr/bin/lsb_release", "-a", 0},
63         {"/usr/bin/free", "-h", 0},
64         {"/usr/bin/df", "-h", "/", 0}
65     };
66
67     #define NUM_OF_CMD (sizeof(cmd)/sizeof(cmd[0]))
68
69     int fd[NUM_OF_CMD][2];
70     for(int i = 0; i < NUM_OF_CMD; i++){
71         myfork(cmd[i], fd[i]);
72     }
73
74     for(int i = 0; i < NUM_OF_CMD; i++){
75         print("\n");
76         print(cmd[i][0]);
77         print(":\n");
78         close(fd[i][1]);
79         printfd(fd[i][0]);
80         close(fd[i][0]);
81     }
82
83     for(int i = 0; i < NUM_OF_CMD; i++){
84         wait(0);
85     }
86
87     return 0;
88 }
```

BÁO CÁO THỰC HÀNH

Sinh viên:

MSSV: Nhóm:

Bài 1: Viết chương trình myecho có chức năng tương tự như echo**Bài 2:** Trả lời các câu hỏi sau:

a) Trong chương trình mycat:

- Hàm mycat ở dòng 20 có chức năng gì? Hàm hoạt động như thế nào?
- Dòng 35 kiểm tra điều gì? Tại sao?
- Dòng 38 tại sao khởi tạo $i = 1$
- Dòng 44 và 45 có chức năng gì?

b) Trong chương trình myls:

- Hàm print ở dòng 32 có chức năng gì?
- Giải thích hoạt động của đoạn code từ dòng 70 đến 84
- Dòng 94 và 95 có chức năng gì?
- Dòng 106 có chức năng gì?
- Dòng 113 có chức năng gì?
- Nếu bỏ dòng 116 thì chương trình bị lỗi gì? Vì sao?

c) Cho biết chương trình myfork chạy đúng hay sai trong các trường hợp sau và giải thích tại sao:

- Bỏ dòng 52 và 53
- Bỏ dòng 78
- Bỏ dòng 80
- Bỏ dòng 84

Bài 3: Viết chương trình mygrep in ra stdout các dòng có chứa từ khóa <K> từ các file text có đường dẫn <P>, với <K> và <P> là các tham số. Nếu không nhận được tham số <P> thì đọc file từ stdin. Tham số <K> có thể chứa kí tự "-" đại diện cho bất kì kí tự đơn nào, kể cả khoảng trắng, nhưng ngoại trừ kí tự xuống dòng

Ví dụ: file1 và file2 có nội dung như sau:

file1	file2
aaa bbb ccc	aabb
aacc bbb aa	c a b
acb	aa
aabc	bb

```
$ ./mygrep -a-b file1 file2
```

```
file1:
aaa bbb ccc
file2:
aabb
c a b
$
```

Bài 4: Viết chương trình myfilecnt nhận tham số <P> là đường dẫn dir và in ra stdout tổng số regular file có trong dir đó và các subdir

Ví dụ:

```
$ ./myfilecnt ~
There are 34 regular files in /home/fetel
$
```

Gợi ý: Đếm các file trong dir hiện tại và thực hiện đếm đệ quy cho các subdir (ngoại trừ các subdir . và ..)

Bài 5: Viết chương trình myshell thực thi command là các chương trình lưu trong thư mục /usr/bin. Nếu không tìm được chương trình thì báo lỗi qua stderr

Ví dụ:

```
$ ./myshell cat ~/date
Fri 23 Sep 2022 10:15:04 AM GMT
$ ./myshell invalid
error: invalid not found in /usr/bin
$
```

Bài 6: Viết chương trình myps in ra stdout thông tin của tất cả các process đang chạy trong hệ thống. Mỗi process là 1 dòng, cột 1 là process PID cột 2 là process command

Ví dụ:

```
$ ./mysps
PID      CMD
1963     /usr/bin/bash
...
$
```

Gợi ý: Dir /proc chứa thông tin của tất cả process đang chạy

Bài 7*: Viết chương trình mytree có chức năng tương tự như command tree, nhận argument <P> là đường dẫn của dir và in ra stdout nội dung dir đó và các subdir dưới dạng cây thư mục

Ví dụ: command `./mytree ~` sẽ in ra Terminal

```

/home/fetel
├── a.out
├── date
├── lab1
│   ├── host.txt
│   ├── mydir
│   ├── myhost
│   └── new
├── list.txt
└── test
    ├── e1
    ├── e2
    ├── e3
    ├── f1
    ├── f2
    ├── f21
    ├── f22
    ├── f3
    ├── f4
    └── new.c

```

Gợi ý: sử dụng kiểu dữ liệu tree (cấp phát động) để lưu trữ nội dung dir hiện tại và thực hiện đệ quy cho các subdir (ngoại trừ các subdir . và ..)

Bài 8*:** Viết chương trình mybash thực hiện chức năng cơ bản của một shell:

- Prompt là kí tự >>
- Thực thi command là các chương trình lưu trong thư mục /usr/bin
- Hỗ trợ đường dẫn bắt đầu bằng ~
- Hỗ trợ các tính năng wildcard (* và ?), stdout/stdin redirection, pipeline tối đa 2 process
- Kết thúc thực thi khi nhận được build-in command exit

```

$ ./myshell
>>
>>
>> echo hello world
hello world
>> cat /home/fetel/date
Fri 23 Sep 2022 10:15:04 AM GMT
>> cat ~/*file.??? | grep zip > ./out
>> exit
Goodbye !
$

```