Lab 2: Lập trình I/O trên Linux

1 Lý thuyết

1.1 Làm việc với file trên Linux

1.1.1 File và các thuộc tính

Trên hệ điều hành Linux, file là một tập hợp dữ liệu được đặt tên lưu trữ trong ổ đĩa. Các loại file khác nhau bao gồm:

- Regular file: là loại file thực sự lưu trữ dữ liệu và có thể được đọc (read), ghi (write) hay thực thi (execute) tùy vào quyền truy cập được cho phép (allowed permission).
- Directory: là loại file đặc biệt có khả năng chứa (contain) các file khác hoặc các subdirectories khác. Chúng ánh xạ (map) tên file với các inode tương ứng (inode là cấu trúc dữ liệu của hệ thống lưu trữ thuộc tính và địa chỉ của file).
  Ta có thể dùng câu lệnh `ls -il` để hiển thị các inode với mỗi inode được gắn với một số nguyên, ví dụ:

```
9175045 drwxr-xr-x 2 Chiaki Chiaki     4096 Mar  1 23:48  Autumn
8914106 -rw-r--r-- 1 Chiaki Chiaki  2152597 Mar 31 19:20  C-Tut-4.02.pdf
```

- Device files: là loại file đặc biệt được hoạt động như cầu nối giữa phần mềm đến phần cứng. Thay vì lưu trữ dữ liệu, nó cho phép các phần mềm được tương tác (communicate) với các device drives tương ứng trong kernel của máy tính, sau đó kernel sẽ tương tác (interact) đến phần cứng. Những file này được lưu trữ tại directory `\dev`.
- FIFO (pipe): là cách thức để các tiến trình (process) tương tác với những tiến trình khác. Nó cho phép một tiến trình được ghi (write) dữ liệu với tiến trình khác đọc (read) chúng theo hình thức FIFO (first-in-first-out).
- Sockets: giống như FIFOs nhưng linh hoạt hơn, có thể tương tác giữa các tiến trình theo hai chiều (bidirectional communication).

Trong Linux, "Everything is a file" là một nguyên lý cơ sở có nghĩa là gần như tất cả các đối tượng (object) mà người dùng tương tác trong hệ thống đều được biểu diễn dưới dạng file và được truy cập sử dụng giao diện file (file-like interface)
⇒ Điều này có nghĩa là khi người dùng sử dụng bất kì một loại file nào thì các câu lệnh giống nhau (same set of system calls command) như mở (open), đọc (read), ghi (write) và đóng (close) đều có thể được áp dụng qua thao tác I/O (I/O operations).
Các file trong Linux không chỉ lưu trữ dữ liệu mà còn có rất nhiều mô tả thuộc tính cụ thể (descriptive attributes) được lưu trữ trong inode của chúng. Người dùng có thể truy cập chúng thông qua `struct stat` là một cấu trúc dữ liệu được sử dụng cho việc lưu giữ các thuộc tính của file trong inode.

```
struct stat{
    ...
    mode_t st_mode; /* File types and permission */
    off_t st_size; /* Total file size (bytes) */
    ...
    }
```

Ví dụ về cách sử dụng các fields (vùng) trong `struct stat` như sau:

1. `st_mode`: đây là field có chức năng xác định loại file (file type) và quyền truy cập (file permission). Thông tin về loại file có thể được truy cập bằng cách sử dụng toán tử AND với hằng số `S_IFMT`, kết quả trả về có thể được so sánh với một khoảng hằng số (range of constants) để xác định loại file:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
int main(void){
    char pathname[] = "/home/Chiaki/LaTeX/fourier.pdf";
    struct stat statbuff;
    if(stat(pathname, &statbuff)==-1){ //Getting the attributes of pathname
        perror("stat");
    }
    if((statbuff.st_mode & S_IFMT)==S_IFREG){ //Syntax not using macro
        printf("Regular file\n");
    }
    if(S_ISREG(statbuff.st_mode)){
        printf("Regular file\n"); //Equivalent syntax using S_ISREG macro
    }
    return 0;
}
```

Tương tự, ta cũng có thể kiểm tra link bất kì có phải là directory hay không khi thay đổi cú pháp và macro:

```
    if((statbuff.st_mode & S_IFMT)==S_IFDIR){ //Syntax not using macro
        printf("Directory\n");
    }
    if(S_ISDIR(statbuff.st_mode)){
        printf("Directory\n"); //Equivalent syntax using S_ISDIR macro
    }
```

2. `st_size`: đây là field có chức năng trả về kích thước của file tính theo đơn vị bytes:

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

int main(void){
```

```
    char pathname[] = "/home/Chiaki/Documents/DSP.pdf";
    struct stat statbuff;
    if(stat(pathname, &statbuff)==-1){
        perror("stat");
    }
    printf("File size: %lld bytes\n",(long long)statbuff.st_size);
    return 0;
}
```

1.1.2 Các thao tác đơn giản với file

1. Mở file (open): Cú pháp lệnh `open()` system call như sau:

```
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags, .../*mode_t*/){
    Return file descriptors on success, -1 on error
}
```

Đối số `flag` (flag argument) được chia làm ba nhóm: truy cập file (access), khởi tạo (create) và trạng thái (open status).

Flag	Purpose
`O_RDONLY`	Open for reading only
`O_WRONLY`	Open for writing only
`O_RDWR`	Open for reading and writing
`O_CREAT`	Create file if it doesn't already exist
`O_APPEND`	Writes are always appened to end of file

Ví dụ:

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int fd = open("/home/Chiaki/LaTeX/fourier.pdf",O_WRONLY);
    if(fd<0){
        perror("open");
        exit(EXIT_FAILURE);
    }
    printf("File opened successfully!");
    return 0;
}
```

2. Đọc file (read): Cú pháp lệnh `read` system call như sau:

```
#include <unistd.h>
ssize_t read(int fd, void *buffer, size_t count);
Returns number of bytes read, 0 on EOF or -1 on error
```

Câu lệnh `read()` đọc dữ liệu từ một file được tham chiếu (referred) bởi `fd`, đối số (argument) `buffer` cung cấp bộ nhớ để lưu trữ dữ liệu đầu vào, đối số `count` cho biết số lượng bytes tối đa có thể đọc. Ví dụ:

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void){
    int fd = open("/home/Chiaki/C/file2.txt",O_RDONLY);
    if(fd<0){
        perror("open");
        exit(EXIT_FAILURE);
        close(fd);
    }
    char buffer[50];
    ssize_t bytesRead = read(fd, buffer, sizeof(buffer)-1);
    if(bytesRead == -1){
        perror("read");
    }
    buffer[bytesRead] = '\0'; //Null-terminate the string
    printf("Read %zd bytes: %s\n",bytesRead,buffer);
    return 0;
}
```

3. Viết file (write): Cú pháp lệnh `write` system call như sau:

```
#include <unistd.h>
ssize_t write(int fd, void *buffer, size_t count);
Return number of bytes written, or -1 on error
```

Câu lệnh `write()` cũng có cấu trúc tương đồng với câu lệnh `read()`, `buffer` là địa chỉ của dữ liệu được ghi, `count` là số bytes được viết từ `buffer`, và `fd` là tham chiếu (referred) đến file được ghi. Ví dụ:

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main (void){
    int fd = open("/home/Chiaki/C/file2.txt",O_RDWR);
    if(fd<0){
        perror("open");
        exit(EXIT_FAILURE);
        close(fd);
    }
    const char *message = "Hello, world\n";
```

```
    ssize_t bytesWritten = write(fd, message, strlen(message));
    if(bytesWritten<0){
        perror("write");
        close(fd);
        exit(EXIT_FAILURE);
    }
    printf("Wrote %zd bytes to file2.txt\n", bytesWritten);
    close(fd); //Close file command
    return 0;
}
```

4. Đóng file (close): Cú pháp lệnh `close` system call như sau:

```
#include <unistd.h>
close(fd);
Return 0 on success, or -1 on error
```

Câu lệnh `close()` dùng để đóng một file sau khi đã sử dụng xong, với `fd` là tham chiếu đến file cần phải đóng. Ví dụ:

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main (void){
int fd = open("/home/Chiaki/C/file2.txt",O_RDWR);
if(fd<0){
    perror("open");
    exit(EXIT_FAILURE);
    close(fd);
    }
return 0;
}
```

1.2 Làm việc với directory

1.2.1 Cấu trúc struct dirent

- Như đã đề cập ở mục 1.1.1, directory trong Linux là một loại file đặc biệt có khả năng chứa các file hoặc directory khác.

- Để thực hiện các thao tác (operation) với directory, tương tự với file như đã đề cập ở cuối mục 1.1.1, ta cũng sử dụng cấu trúc `dirent` (giống với `stat`).

- Cấu trúc `struct dirent` lưu giữ các thông tin của directory như là `d_ino`: mã số inode hay `d_name`: tên của entry.

```
struct dirent{
    ...
    ino_t d_ino;
    char d_name[];
    ...
}
```

1.2.2 Các thao tác đơn giản với directory

1. Mở directory (open): Cú pháp lệnh `opendir()` system call như sau:

```
#include <dirent.h>
DIR *opendir(const char *dirpath);
// Return NULL if directory can't be opened
```

Ví dụ:

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

int main(void){
    DIR *dp = opendir("/home/Chiaki/C");
    if(dp==NULL){
        perror("opendir");
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

2. Đọc directory (read): Cú pháp lệnh `readdir()` system call như sau:

```
#include <dirent.h>
struct dirent *readdir(DIR *dirp);
Return pointer to a statically allocated structure describing next directory
entry, or NULL on EOD or error
```

Ví dụ:

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

int main(void){
    DIR *dp = opendir("/home");
    if(dp==NULL){
        perror("open");
        exit(EXIT_FAILURE);
    }
```

```
    struct dirent *entry;
    while((entry=readdir(dp))!=NULL){
        printf("Found entry: %s\n", entry->d_name);
    }
    return 0;
}
```

3. Tạo directory (make directory): Cú pháp lệnh `mkdir()` system call như sau:

```
#include <sys/stat.h>
int mkdir(const char *pathname, mode_t mode);
// Return 0 on success, or -1 on error
```

   (a) Đối số (argument) `pathname` xác định đường dẫn (directory) mới được khởi tạo, đường dẫn này có thể là đường dẫn tương đối (relative) hoặc tuyệt đối (absolute). Nếu một file với đường dẫn này đã tồn tại thì lệnh `mkdir()` sẽ trả về lỗi `EEXIST`.

   (b) Đối số `mode` xác định quyền truy cập (permission), ví dụ như `0755` cho phép người dùng (user) có toàn quyền truy cập, trong khi các nhóm (groups) hay người khác (others) chỉ được quyền đọc (read) và ghi (write).
   Ví dụ:

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

int main(void){
    if(mkdir("/home/Chiaki/C",0755)==-1){
        perror("mkdir");
        exit(EXIT_FAILURE);
    }
    printf("Directory created successfully!\n");
    return 0;
}
```

4. Xóa directory (remove directory): Cú pháp lệnh `rmdir()` system call như sau:

```
#include <unistd.h>
int rmdir(const char *pathname);
Return 0 on success, -1 on failure
```

   Lưu ý, để `rmdir()` thành công thì directory phải **rỗng**.
   Ví dụ:

```
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(void){
```

```
    int res = mkdir("/home/Chiaki/C/path", S_IRUSR | S_IXUSR);
    if(res == 0){
        printf("The directory was created\n");
    }
    res = rmdir("/home/Chiaki/C/path");
    if(res == 0){
        printf("The directory was removed");
    }
    return 0;
}
```

2 Thực hành

2.1 Làm việc với file

2.1.1 Exercise 1:

1.
```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>

int main(void){
    int fd = open("/home/Chiaki/C/Hello.txt",O_RDONLY);
    if(fd<0){
        perror("open");
        exit(EXIT_FAILURE);
        close(fd);
    }
    struct stat statbuff;
    if(stat("/home/Chiaki/C/Hello.txt", &statbuff)==-1){
        perror("stat");
        close(fd);
    }
    long long MAX_READ = statbuff.st_size;
    char buffer[MAX_READ + 1];
    ssize_t bytesRead = read(fd, buffer, sizeof(buffer)-1);
    if(bytesRead == -1){
        perror("read");
        close(fd);
    }
    buffer[MAX_READ] = '\0';
    printf("Read %zd bytes: %s\n", bytesRead, buffer);
    close(fd);
    return 0;
}
```

2.
```
        if (lseek(fd,6,SEEK_SET)==-1) {
            perror("lseek");
```

```
        close(fd);
        return 1;
    }

    char buffer[BUFFER_SIZE];
    ssize_t bytesRead = read(fd, buffer, sizeof(buffer)-1);

    if (bytesRead==-1) {
        perror("read");
        close(fd);
        return 1;
    }
```

2.1.2 Exercise 2:

1. Đọc file với input được nhập từ keyboard:

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main(void){
    int fd = open("/home/Chiaki/C/text.txt",O_CREAT | O_RDWR);
    if(fd < 0){
        perror("open");
        exit(EXIT_FAILURE);
    }
    char buffer[256];
    printf("Input from keyboard: ");
    fgets(buffer, sizeof(buffer), stdin);
    ssize_t bytesWritten = write(fd, buffer, strlen(buffer));
    if(bytesWritten < 0){
        perror("write");
        close(fd);
        exit(EXIT_FAILURE);
    }
    close(fd);
    return 0;
}
```

2. Đọc file với input từ một file khác:

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
int main(void){
    int fd = open("/home/Chiaki/C/text.txt",O_CREAT | O_RDWR);
```

```
            if(fd < 0){
                perror("open");
                exit(EXIT_FAILURE);
            }
            int fd2 = open("/home/Chiaki/C/Hello.txt", O_RDONLY);
            if(fd2 < 0){
                perror("open");
                exit(EXIT_FAILURE);
            }
            char buffer[124];
            ssize_t bytesRead = read(fd2, buffer, sizeof(buffer)-1);
            ssize_t bytesWritten = write(fd, buffer, strlen(buffer));
            if(bytesWritten < 0){
                perror("write");
                close(fd);
                exit(EXIT_FAILURE);
            }
            close(fd);
            return 0;
        }
```

2.1.3 Exercise 3:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <stdlib.h>
int main(void){
    printf("Enter the name of a file: ");
    char pathname[256];
    scanf("%s",pathname);
    struct stat statbuff;
    if(stat(pathname, &statbuff)==-1){
        perror("stat");
    }
    printf("File size: %lld bytes\n", (long long)statbuff.st_size);
    printf("User permission: ");
    if((statbuff.st_mode & S_IRUSR)!=-1){
        printf("-r");
    }
    if((statbuff.st_mode & S_IWUSR)!=-1){
        printf("w");
    }
    if((statbuff.st_mode & S_IXUSR)!=-1){
        printf("x");
    }
    return 0;
}
```

2.1.4 Exercise 4:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

int main(void){
    int fd = open("/home/Chiaki/C/example.txt",O_CREAT | O_RDWR);
    if(fd<0){
        perror("open");
        exit(EXIT_FAILURE);
        close(fd);
    }
    printf("Input string you want to append: ");
    char string[256];
    scanf("%s",string);
    ssize_t bytesWritten =write(fd, string, strlen(string));
    if(bytesWritten < 0){
        perror("write");
        close(fd);
        exit(EXIT_FAILURE);
    }
    close(fd);
    return 0;
}
```

2.1.5 Exercise 5:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFFER_SIZE 1024

void grep(const char *pattern, const char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("Error opening file");
        return;
    }

    char buffer[BUFFER_SIZE];
    while (fgets(buffer, BUFFER_SIZE, file) != NULL) {
        if (strstr(buffer, pattern) != NULL) {
            printf("%s", buffer);
        }
    }

    fclose(file);
```

```
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]);
        return EXIT_FAILURE;
    }

    const char *pattern = argv[1];
    const char *filename = argv[2];

    grep(pattern, filename);

    return EXIT_SUCCESS;
}
```

2.2 Làm việc với directory

2.2.1 Exercise 1:

```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(void){
    char buffer[1000];
    scanf("%s", buffer);
    DIR *dp = opendir(buffer);
    if(dp==NULL){
        perror("open dir");
        exit(EXIT_FAILURE);
    }
    struct dirent *entry;
    while((entry=readdir(dp))!=NULL){
        printf("Entry: %s\n", entry->d_name);
    }
    return 0;
}
```

2.2.2 Exercise 2:

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

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <directory>\n", argv[0]);
        return 1;
```

```
    }
    if (chdir(argv[1]) != 0) {
        perror("chdir");
        return 1;
    }
    char cwd[1024];
    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        printf("Current directory: %s\n", cwd);
    } else {
        perror("getcwd");
        return 1;
    }
    return 0;
}
```