

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Lê Hoài Nghĩa.

Họ và tên	MSSV	Lớp
Lại Quan Thiên	22521385	IT007.O211.1
Đặng Đức Tài	22521270	
Mai Nguyễn Nam Phương	22521164	
Phùng Trần Thế Nam	21522366	

HỆ ĐIỀU HÀNH BÁO CÁO LAB 6

CHECKLIST

6.4. BÀI TẬP THỰC HÀNH

	Câu 1	Câu 2	Câu 3	Câu 4	Câu 5
Trình bày giải thuật	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Chụp hình minh chứng (chạy ít nhất 3 lệnh)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Giải thích code, kết quả	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Tự chấm điểm: 10/10

**Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:
<Tên nhóm>_LAB6.pdf*

6.4. BÀI TẬP THỰC HÀNH

* FULL SOURCE CODE:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/wait.h>
4  #include <string.h>
5  #include <signal.h>
6  #include <fcntl.h>
7  #include <sys/types.h>
8  #include <stdlib.h>
9
10 #define MAX_LINE 80
11 #define HISTORY_COUNT 10 // Số lượng lệnh lịch sử tối đa
12 pid_t pid;
13
14 void on_sigint(){
15     printf("\nCtrl C was pressed\n");
16     kill(pid, SIGINT);
17 }
18
19 char history[HISTORY_COUNT][MAX_LINE]; // Mảng lưu trữ lịch sử lệnh
20 int history_count = 0; // Số lượng lệnh đã lưu trữ trong lịch sử
21
22 void add_to_history(const char *input) {
23     if (history_count < HISTORY_COUNT) {
24         strcpy(history[history_count++], input);
25     } else {
26         // Nếu mảng lịch sử đầy, xóa lệnh đầu tiên và thêm lệnh mới vào cuối
27         for (int i = 1; i < HISTORY_COUNT; i++) {
28             strcpy(history[i - 1], history[i]);
29         }
30         strcpy(history[HISTORY_COUNT - 1], input);
31     }
32 }
```

```
34 void print_history() {
35     printf("Command history:\n");
36     for (int i = history_count - 1; i >= 0; i--) {
37         printf("%d: %s\n", history_count - i - 1, history[i]);
38     }
39 }
40
41 int main(void) {
42     char *args[MAX_LINE];
43     int should_run = 1;
44     char input[MAX_LINE];
45     char *args_pipe[MAX_LINE];
46
47     while (should_run){
48         signal(SIGINT, on_sigint);
49         printf("it007sh>");
50         fflush(stdout);
51
52         fgets(input, MAX_LINE, stdin);
53         input[strlen(input) - 1] = '\0';
54
55         if (strcmp(input, "exit") == 0){
56             should_run = 0;
57             break;
58         }
59
60         if (strcmp(input, "history") == 0) {
61             print_history();
62             continue;
63         }
```

```
65     if (strcmp(input, "HF") == 0) {
66         if (history_count > 0) {
67             print_history();
68             continue;
69         } else {
70             printf("No commands in history.\n");
71             continue;
72         }
73     }
74
75     add_to_history(input); // Thêm lệnh vào lịch sử
76
77     char *token = strtok(input, " ");
78     int i = 0;
79     while (token != NULL){
80         args[i++] = token;
81         token = strtok(NULL, " ");
82     }
83     args[i] = NULL;
84
85     int chuyenhuongdauvao = 0, chuyenhuongdaura = 0, pipe_idx = -1;
86     char *inputFile = NULL, *outputFile = NULL;
87     for (int i = 0; args[i] != NULL; i++){
88         if (strcmp(args[i], "<") == 0){
89             chuyenhuongdauvao = 1;
90             inputFile = args[i + 1];
91             args[i] = NULL;
92         }
93         else if (strcmp(args[i], ">") == 0){
94             chuyenhuongdaura = 1;
95             outputFile = args[i + 1];
```

```
96     args[i] = NULL;
97 }
98 else if (strcmp(args[i], "|") == 0){
99     pipe_idx = i;
100    args[i] = NULL;
101    int j = 0;
102    for (int m = i+1; args[m] != NULL; m++){
103        args_pipe[j++] = args[m];
104        args[m] = NULL;
105    }
106    args_pipe[j] = NULL;
107    break;
108 }
109 }
110 pid = fork();
111 if (pid < 0){
112     printf("Failed to fork.\n");
113     exit(1);
114 }
115 else if (pid == 0){
116     if (chuyenhuongdauvao){
117         int fd = open(inputFile, O_RDONLY);
118         dup2(fd, STDIN_FILENO);
119         close(fd);
120     }
121     if (chuyenhuongdaura){
122         int fd = open(outputFile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
123         dup2(fd, STDOUT_FILENO);
124         close(fd);
125     }
126     if (pipe_idx != -1){
```

```
127         int fd[2];
128         pipe(fd);
129         pid_t pid2 = fork();
130         if (pid2 == 0){
131             close(fd[0]);
132             dup2(fd[1], STDOUT_FILENO);
133             close(fd[1]);
134             execvp(args[0], args);
135             exit(1);
136         }
137         else{
138             close(fd[1]);
139             dup2(fd[0], STDIN_FILENO);
140             close(fd[0]);
141             execvp(args_pipe[0], args_pipe);
142             exit(1);
143         }
144     }
145     else{
146         execvp(args[0], args);
147         perror("execvp failed");
148         exit(1);
149     }
150 }
151 else
152     wait(NULL);
153 }
154 return 0;
155 }
156
```

1. Câu 1

CODE:

```
pid = fork();
if (pid < 0){
    printf("Failed to fork.\n");
    exit(1);
}
else if (pid == 0){
    if (chuyenhuongdauvao){
        int fd = open(inputFile, O_RDONLY);
        dup2(fd, STDIN_FILENO);
        close(fd);
    }
    if (chuyenhuongdaura){
        int fd = open(outputFile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        dup2(fd, STDOUT_FILENO);
        close(fd);
    }
    if (pipe_idx != -1){
        int fd[2];
        pipe(fd);
        pid_t pid2 = fork();
        if (pid2 == 0){
            close(fd[0]);
            dup2(fd[1], STDOUT_FILENO);
            close(fd[1]);
            execvp(args[0], args);
            exit(1);
        }
        else{
            close(fd[1]);
            dup2(fd[0], STDIN_FILENO);
            close(fd[0]);
            execvp(args_pipe[0], args_pipe);
            exit(1);
        }
    }
    else{
        execvp(args[0], args);
        perror("execvp failed");
        exit(1);
    }
}
else
    wait(NULL);
```

GIẢI THÍCH CODE:

1. Khởi tạo tiến trình con bằng fork(): Hàm fork() tạo ra một tiến trình con từ tiến trình cha. Nếu fork() trả về giá trị âm, điều này có nghĩa là tiến trình cha không thể tạo được tiến trình con, và chương trình sẽ kết thúc với thông báo lỗi.

2. Thực thi lệnh trong tiến trình con: Nếu fork() trả về 0, điều này có nghĩa là mã đang chạy trong tiến trình con. Tiến trình con sẽ thực thi lệnh được nhập bởi người dùng.

3. Chuyển hướng đầu vào: Nếu lệnh yêu cầu chuyển hướng đầu vào (kí hiệu <), tiến trình con sẽ mở tệp chỉ định ở chế độ chỉ đọc và thay thế đầu vào chuẩn (stdin) bằng mô tả tệp của tệp này. Điều này có nghĩa là dữ liệu từ tệp sẽ được dùng làm đầu vào cho lệnh.

4. Chuyển hướng đầu ra: Nếu lệnh yêu cầu chuyển hướng đầu ra (kí hiệu >), tiến trình con sẽ mở hoặc tạo tệp chỉ định ở chế độ ghi, và thay thế đầu ra chuẩn (stdout) bằng mô tả tệp của tệp này. Điều này có nghĩa là dữ liệu đầu ra của lệnh sẽ được ghi vào tệp thay vì hiển thị trên màn hình.

5. Sử dụng pipe (|): Nếu lệnh chứa kí hiệu pipe (|), tiến trình con sẽ tạo một tiến trình con mới để thực thi lệnh trước kí hiệu pipe và chuyển hướng đầu ra của lệnh này đến đầu vào của lệnh sau pipe. Pipe cho phép kết nối đầu ra của một lệnh với đầu vào của lệnh khác.

6. Thực thi lệnh: Tiến trình con sẽ thực thi lệnh bằng cách sử dụng hàm execvp(), thay thế nội dung của tiến trình con bằng lệnh mới. Nếu execvp() thất bại, chương trình sẽ in ra thông báo lỗi và thoát.

7. Tiến trình cha đợi tiến trình con hoàn thành: Sau khi tạo tiến trình con, tiến trình cha sẽ đợi tiến trình con hoàn thành trước khi tiếp tục vòng lặp để đọc và thực thi lệnh

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Lê Hoài Nghĩa.

tiếp theo. Điều này đảm bảo rằng shell sẽ không chấp nhận lệnh mới từ người dùng cho đến khi lệnh hiện tại hoàn thành.

CHẠY CODE:

```
thenam@21522366:~/Documents$ gcc -o Lab_6 Lab_6.c
thenam@21522366:~/Documents$ ./Lab_6
it007sh>echo hello > hello.txt
it007sh>cat hello.txt
hello
it007sh>ls | grep hello
hello
hello.txt
it007sh>exit
thenam@21522366:~/Documents$
```

1. Tạo và ghi vào tệp hello.txt: `echo hello > hello.txt`

-> Tạo tệp hello.txt và ghi chuỗi "hello" vào đó.

2. Hiển thị nội dung của hello.txt: `cat hello.txt`

-> Hiển thị nội dung của tệp hello.txt, kết quả là "hello".

3. Liệt kê và tìm kiếm các tệp có tên chứa "hello": `ls | grep hello`

-> Liệt kê các tệp và thư mục trong thư mục hiện tại, rồi tìm và hiển thị các tên có chứa "hello".

2. Câu 2

CODE:

```
char history[HISTORY_COUNT][MAX_LINE]; // Mảng lưu trữ lịch sử lệnh
int history_count = 0; // Số lượng lệnh đã lưu trữ trong lịch sử

void add_to_history(const char *input) {
    if (history_count < HISTORY_COUNT) {
        strcpy(history[history_count++], input);
    } else {
        // Nếu mảng lịch sử đầy, xóa lệnh đầu tiên và thêm lệnh mới vào cuối
        for (int i = 1; i < HISTORY_COUNT; i++) {
            strcpy(history[i - 1], history[i]);
        }
        strcpy(history[HISTORY_COUNT - 1], input);
    }
}

void print_history() {
    printf("Command history:\n");
    for (int i = history_count - 1; i >= 0; i--) {
        printf("%d: %s\n", history_count - i - 1, history[i]);
    }
}
```

GIẢI THÍCH:

- Đầu tiên ta sẽ tạo mảng 'history' để lưu trữ các lệnh đã thực thi.
- Tạo 1 biến 'history_count' để giữ số lượng lệnh trong lịch sử.
- Hàm 'add_to_history' dùng để ta push các lệnh đã nhập vào trong mảng lưu, nếu mảng đã đầy thì ta sẽ xóa lệnh đầu tiên và thêm lệnh mới vào
- Hàm 'print_history' sẽ cho ta xem các lệnh đã được ta nhập (lưu trong mảng history)

```
if (strcmp(input, "HF") == 0) {
    if (history_count > 0) {
        print_history();
        continue;
    } else {
        printf("No commands in history.\n");
        continue;
    }
}

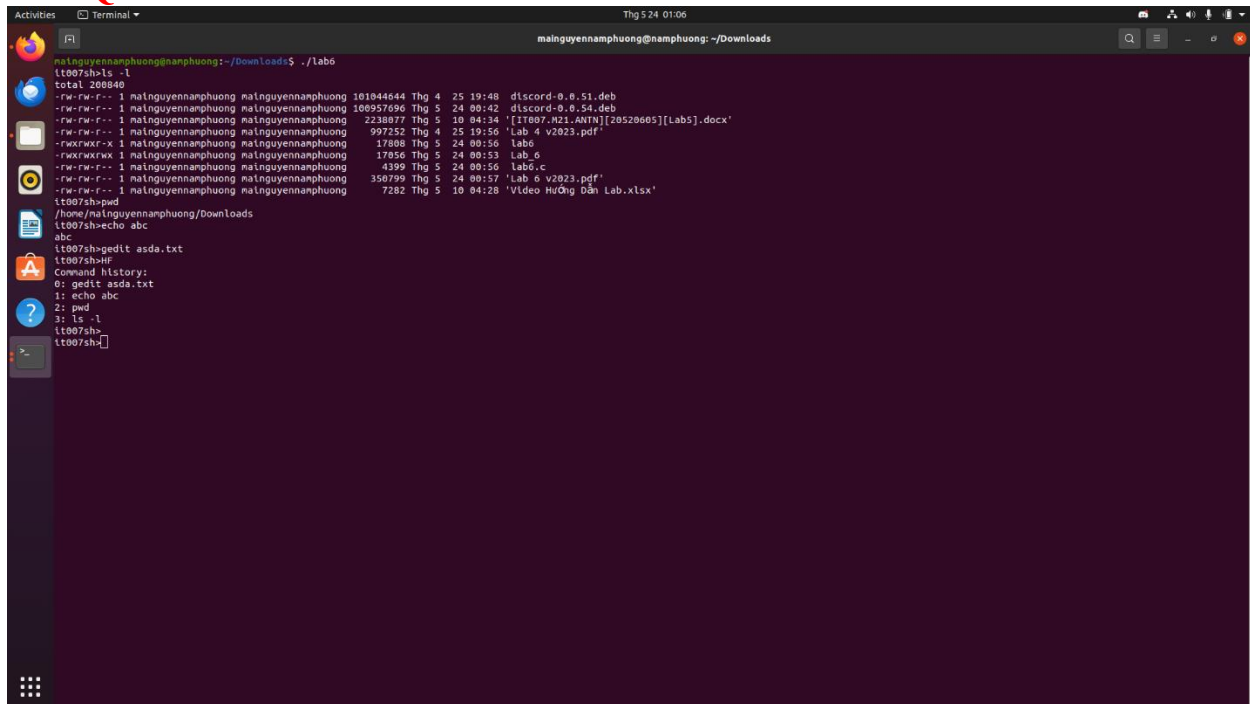
add_to_history(input); // Thêm lệnh vào lịch sử
```

- Đây là hàm xử lý trong hàm main, dễ dàng nhận thấy lệnh sau khi được nhập sẽ được lưu lại nhờ hàm add_to_history(), sau đó nếu input của người dùng là "HF" thì ta sẽ gọi kiểm tra biến history_count, nếu nó > 0 thì chứng tỏ đã có lệnh được

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Lê Hoài Nghĩa.

thực hiện, khi đó ta sẽ gọi hàm 'print_history' nhằm cho người dùng xem lịch sử các câu lệnh

KẾT QUẢ:



```
mainguyennamphuong@namphuong: ~/Downloads
mainguyennamphuong@namphuong:~/Downloads$ ./lab6
total 208840
-rw-rw-r-- 1 mainguyennamphuong mainguyennamphuong 161844444 Thg 4 25 19:48 discord-0.0.51.deb
-rw-rw-r-- 1 mainguyennamphuong mainguyennamphuong 100957096 Thg 5 24 00:42 discord-0.0.54.deb
-rw-rw-r-- 1 mainguyennamphuong mainguyennamphuong 2238077 Thg 5 10 04:34 '[IT007_R21.ANTN][20520605][Lab5].docx'
-rw-rw-r-- 1 mainguyennamphuong mainguyennamphuong 997252 Thg 4 25 19:56 'Lab 4 v2023.pdf'
-rwxrwxr-x 1 mainguyennamphuong mainguyennamphuong 17688 Thg 5 24 00:56 Lab 6
-rwxrwxr-x 1 mainguyennamphuong mainguyennamphuong 17056 Thg 5 24 00:53 Lab 6
-rw-rw-r-- 1 mainguyennamphuong mainguyennamphuong 4399 Thg 5 24 00:56 Lab 6.c
-rw-rw-r-- 1 mainguyennamphuong mainguyennamphuong 350799 Thg 5 24 00:57 'Lab 6 v2023.pdf'
-rw-rw-r-- 1 mainguyennamphuong mainguyennamphuong 7282 Thg 5 10 04:28 'Video Hướng dẫn Lab.Xlsx'
mainguyennamphuong@namphuong:~/Downloads$ cd /home/mainguyennamphuong/Downloads
mainguyennamphuong@namphuong:~/Downloads$ ls
abc
mainguyennamphuong@namphuong:~/Downloads$ cat asda.txt
abc
mainguyennamphuong@namphuong:~/Downloads$ cat asda.txt
0: gedit asda.txt
1: echo abc
2: pwd
3: ls -l
mainguyennamphuong@namphuong:~/Downloads$
```

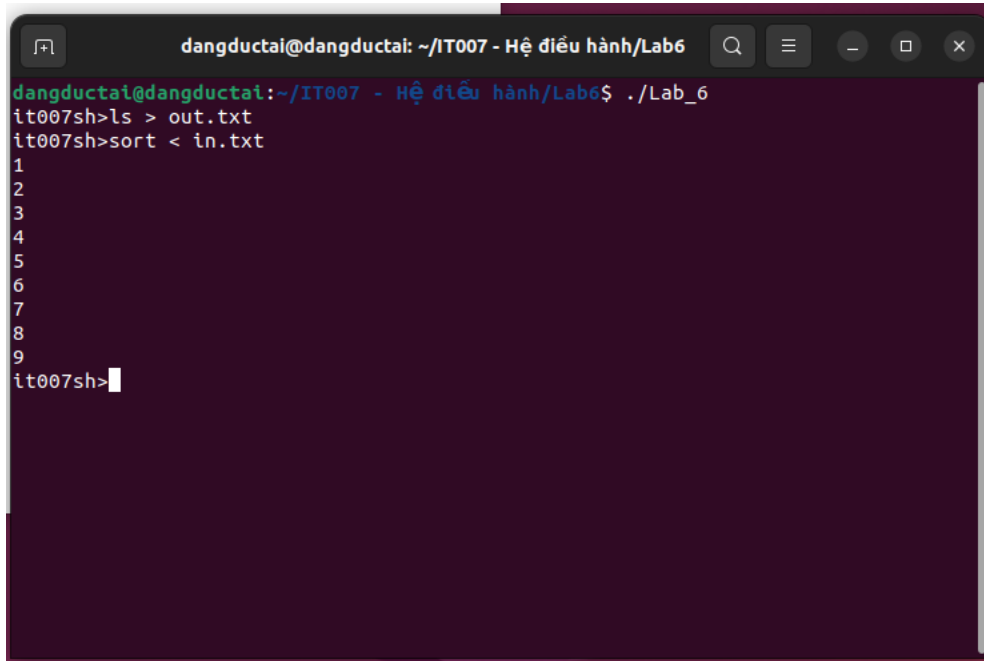
3. Câu 3

CODE:

```
int chuyenhuongdauvao = 0, chuyenhuongdaura = 0, pipe_idx = -1;
char *inputFile = NULL, *outputFile = NULL;
for (int i = 0; args[i] != NULL; i++){
    if (strcmp(args[i], "<") == 0){
        chuyenhuongdauvao = 1;
        inputFile = args[i + 1];
        args[i] = NULL;
    }
    else if (strcmp(args[i], ">") == 0){
        chuyenhuongdaura = 1;
        outputFile = args[i + 1];
        args[i] = NULL;
    }
    else if (strcmp(args[i], "|") == 0){
        pipe_idx = i;
        args[i] = NULL;
        int j = 0;
        for (int m = i+1; args[m] != NULL; m++){
            args_pipe[j++] = args[m];
            args[m] = NULL;
        }
        args_pipe[j] = NULL;
        break;
    }
}

pid = fork();
if (pid < 0){
    printf("Failed to fork.\n");
    exit(1);
}
else if (pid == 0){
    if (chuyenhuongdauvao){
        int fd = open(inputFile, O_RDONLY);
        dup2(fd, STDIN_FILENO);
        close(fd);
    }
    if (chuyenhuongdaura){
        int fd = open(outputFile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        dup2(fd, STDOUT_FILENO);
        close(fd);
    }
}
```

KẾT QUẢ:



```
dangductai@dangductai: ~/IT007 - Hệ điều hành/Lab6
dangductai@dangductai:~/IT007 - Hệ điều hành/Lab6$ ./Lab_6
it007sh>ls > out.txt
it007sh>sort < in.txt
1
2
3
4
5
6
7
8
9
it007sh>
```

GIẢI THÍCH:

- Kiểm tra và thiết lập các cờ chuyển hướng:
 - + *chuyenhuongdauvao* và *chuyenhuongdaura* là các cờ (flag) để xác định xem có yêu cầu chuyển hướng đầu vào hoặc đầu ra không.
 - + *inputFile* và *outputFile* lưu trữ tên tệp sẽ được sử dụng cho chuyển hướng đầu vào và đầu ra.
 - + Vòng lặp for duyệt qua các phần tử trong args để tìm các ký tự <, >, và |.
- Nếu < được tìm thấy:
 - + Đặt *chuyenhuongdauvao* thành 1 để đánh dấu rằng có chuyển hướng đầu vào.
 - + Lưu tên tệp đầu vào vào *inputFile*.
 - + Đặt args[i] thành NULL để ngắt chuỗi tại vị trí này.
- Nếu > được tìm thấy:
 - + Đặt *chuyenhuongdaura* thành 1 để đánh dấu rằng có chuyển hướng đầu ra.
 - + Lưu tên tệp đầu ra vào *outputFile*.
 - + Đặt args[i] thành NULL để ngắt chuỗi tại vị trí này.
- Nếu | được tìm thấy:
 - + Đặt *pipe_idx* thành vị trí của |.
 - + Ngắt chuỗi args tại vị trí này bằng cách đặt args[i] thành NULL.
 - + Sao chép các lệnh sau | vào args_pipe.

- Chuyển hướng đầu vào (Nếu *chuyenhuongdauvao* là 1):
 - + Mở tệp *inputFile* với quyền đọc (*O_RDONLY*).
 - + Dùng *dup2(fd, STDIN_FILENO)* để chuyển hướng đầu vào từ tệp này thay vì từ bàn phím (*STDIN_FILENO*).
 - + Đóng tệp mô tả fd sau khi đã chuyển hướng.
- Chuyển hướng đầu ra (Nếu *chuyenhuongdaura* là 1):
 - + Mở tệp *outputFile* với quyền ghi (*O_WRONLY*). Nếu tệp không tồn tại, tạo tệp mới (*O_CREAT*) và xóa nội dung tệp nếu nó đã tồn tại (*O_TRUNC*).
 - + Dùng *dup2(fd, STDOUT_FILENO)* để chuyển hướng đầu ra đến tệp này thay vì màn hình (*STDOUT_FILENO*).
 - + Đóng tệp mô tả fd sau khi đã chuyển hướng.

4. Câu 4

CODE & GIẢI THÍCH:

```
98     else if (strcmp(args[i], "|") == 0){
99         pipe_idx = i;
100        args[i] = NULL;
101        int j = 0;
102        for (int m = i+1; args[m] != NULL; m++){
103            args_pipe[j++] = args[m];
104            args[m] = NULL;
105        }
106        args_pipe[j] = NULL;
107        break;
108    }
```

Ảnh 1

* Ảnh 1:

1. Khởi tạo biến `pipe_idx`: Lưu vị trí của ký tự pipe (|) trong mảng lệnh `args`. Ban đầu gán giá trị -1 để kiểm tra sau này.
2. Duyệt qua mảng `args` để tìm các lệnh đặc biệt:
 - Vòng lặp `for` duyệt qua từng phần tử của `args` (mảng chứa các lệnh và tham số đã tách từ đầu vào của người dùng).
 - Pipe (|):
 - Nếu phần tử hiện tại là |, lưu chỉ mục của pipe vào `pipe_idx`, đặt `args[i]` thành `NULL` để ngắt lệnh tại vị trí đó.
 - Sau đó, sao chép các phần tử sau | vào mảng `args_pipe`.

```
126         if (pipe_idx != -1){
127             int fd[2];
128             pipe(fd);
129             pid_t pid2 = fork();
130             if (pid2 == 0){
131                 close(fd[0]);
132                 dup2(fd[1], STDOUT_FILENO);
133                 close(fd[1]);
134                 execvp(args[0], args);
135                 exit(1);
136             }
137             else{
138                 close(fd[1]);
139                 dup2(fd[0], STDIN_FILENO);
140                 close(fd[0]);
141                 execvp(args_pipe[0], args_pipe);
142                 exit(1);
143             }
144         }
```

Ảnh 2

*** Ảnh 2:**

1. **Kiểm tra lệnh pipe:** Nếu **pipe_idx** khác **-1**, điều này có nghĩa là người dùng đã nhập lệnh có chứa pipe (**|**).
2. **Tạo pipe:** Sử dụng **pipe(fd)** để tạo một pipe. **fd[0]** là đầu đọc, **fd[1]** là đầu ghi.
3. **Tạo tiến trình con thứ hai (pid2):**
 - Gọi **fork()** để tạo tiến trình con thứ hai.
 - Nếu **pid2 == 0** (trong tiến trình con thứ hai):
 - Đóng đầu đọc của pipe (**fd[0]**).
 - Sử dụng **dup2(fd[1], STDOUT_FILENO)** để chuyển đầu ra chuẩn sang đầu ghi của pipe.
 - Đóng đầu ghi của pipe (**fd[1]**).
 - Gọi **execvp(args[0], args)** để thực thi lệnh trước pipe.
 - Nếu lệnh **execvp** thất bại, gọi **exit(1)** để kết thúc tiến trình con.
4. **Trong tiến trình cha:**
 - Đóng đầu ghi của pipe (**fd[1]**).
 - Sử dụng **dup2(fd[0], STDIN_FILENO)** để chuyển đầu vào chuẩn sang đầu đọc của pipe.
 - Đóng đầu đọc của pipe (**fd[0]**).
 - Gọi **execvp(args_pipe[0], args_pipe)** để thực thi lệnh sau pipe.
 - Nếu lệnh **execvp** thất bại, gọi **exit(1)** để kết thúc tiến trình cha.

*** Cách hoạt động của pipe:**

1. **Tạo pipe:** Hệ thống tạo ra hai file descriptors, một cho đầu đọc và một cho đầu ghi.
2. **Tạo tiến trình con thứ hai:** Tiến trình con thứ hai thiết lập để ghi đầu ra của lệnh đầu tiên vào đầu ghi của pipe.
3. **Thiết lập trong tiến trình cha:** Tiến trình cha thiết lập để đọc đầu vào từ đầu đọc của pipe.
4. **Ghi và đọc qua pipe:**
 - Khi lệnh đầu tiên (**ps aux**) ghi dữ liệu vào pipe, dữ liệu đó sẽ được lưu trữ tạm thời trong pipe.
 - Lệnh thứ hai (**grep firefox**) sẽ đọc dữ liệu từ pipe như thể nó đang đọc từ đầu vào chuẩn (**stdin**).

* Ví dụ hoạt động:

Giả sử ta nhập lệnh **ps aux | grep firefox**:

- Lệnh **ps aux** sẽ được thực thi trong tiến trình con thứ hai.
- Đầu ra của **ps aux** sẽ được ghi vào pipe.
- Tiến trình cha đọc đầu vào từ pipe và thực thi lệnh **grep firefox**.
- Kết quả là **grep firefox** sẽ tìm kiếm từ khóa "firefox" trong đầu ra của **ps aux**.

KẾT QUẢ:

```
wan_thinnn@wanthinnn-MacBookAir: ~/Documents/IT007/Lab_6
wan_thinnn@wanthinnn-MacBookAir:~/Documents/IT007/Lab_6$ ls
hello.pem hi.txt in.txt Lab_6 Lab_6.c out.txt
wan_thinnn@wanthinnn-MacBookAir:~/Documents/IT007/Lab_6$ ./Lab_6
tt007sh>ps aux | grep firefox
wan_thi+ 1860  6.1  4.8 11699780 384088 ?        SL  11:16   0:26 /snap/firefox/4209/usr/lib/firefox/firefox
wan_thi+ 2009  0.0  0.5 210636 47872 ?          SL  11:16   0:00 /snap/firefox/4209/usr/lib/firefox/firefox -contentproc -parentBuil
dID 20240427193319 -prefsLen 28823 -prefMapSize 241654 -appDir /snap/firefox/4209/usr/lib/firefox/browser {e3d42f98-f40b-491c-8aa7-39b
4489e9dd2} 1860 true socket
wan_thi+ 2026  0.2  1.4 2490008 118748 ?        SL  11:16   0:01 /snap/firefox/4209/usr/lib/firefox/firefox -contentproc -childID 1
-isForBrowser -prefsLen 28964 -prefMapSize 241654 -jsInitLen 234952 -parentBuildID 20240427193319 -greomni /snap/firefox/4209/usr/lib/
firefox/omni.ja -appomni /snap/firefox/4209/usr/lib/firefox/browser/omni.ja -appDir /snap/firefox/4209/usr/lib/firefox/browser {bd1dc4
5e-ca2c-4217-b2e8-9ba4e639141f} 1860 true tab
wan_thi+ 2135  0.1  1.1 2431236 93880 ?          SL  11:16   0:00 /snap/firefox/4209/usr/lib/firefox/firefox -contentproc -childID 2
-isForBrowser -prefsLen 34454 -prefMapSize 241654 -jsInitLen 234952 -parentBuildID 20240427193319 -greomni /snap/firefox/4209/usr/lib/
firefox/omni.ja -appomni /snap/firefox/4209/usr/lib/firefox/browser/omni.ja -appDir /snap/firefox/4209/usr/lib/firefox/browser {b11a2c
df-0e30-4c8c-94ed-e2fa24e01f4} 1860 true tab
wan_thi+ 2305  0.0  0.5 209340 44800 ?          SL  11:16   0:00 /snap/firefox/4209/usr/lib/firefox/firefox -contentproc -parentBuil
dID 20240427193319 -sandboxingKind 0 -prefsLen 34508 -prefMapSize 241654 -appDir /snap/firefox/4209/usr/lib/firefox/browser {21e9cb3c-
a9e3-4364-b1e8-cd7f46de77ab} 1860 true utility
wan_thi+ 2311  1.8  3.1 6879372 252024 ?        SL  11:16   0:07 /snap/firefox/4209/usr/lib/firefox/firefox -contentproc -childID 3
-isForBrowser -prefsLen 31117 -prefMapSize 241654 -jsInitLen 234952 -parentBuildID 20240427193319 -greomni /snap/firefox/4209/usr/lib/
firefox/omni.ja -appomni /snap/firefox/4209/usr/lib/firefox/browser/omni.ja -appDir /snap/firefox/4209/usr/lib/firefox/browser {a11f41
77-2abd-4f88-8ee9-c4dc342151e4} 1860 true tab
wan_thi+ 2319  0.8  2.1 2518192 168332 ?        SL  11:16   0:03 /snap/firefox/4209/usr/lib/firefox/firefox -contentproc -childID 5
-isForBrowser -prefsLen 31117 -prefMapSize 241654 -jsInitLen 234952 -parentBuildID 20240427193319 -greomni /snap/firefox/4209/usr/lib/
firefox/omni.ja -appomni /snap/firefox/4209/usr/lib/firefox/browser/omni.ja -appDir /snap/firefox/4209/usr/lib/firefox/browser {9d5495
47-7eee-4738-9a4b-c1d9cfcb19be} 1860 true tab
```

5. Câu 5

CODE:

```
pid_t pid;

void on_sigint(){
    printf("\nCtrl C was pressed\n");
    kill(pid, SIGINT);
}
```

- Khi người dùng nhấn Ctrl+C, hàm on_sigint sẽ được gọi. Trong hàm này:
 - In ra thông báo “Ctrl C was pressed”.
 - Gửi tín hiệu SIGINT đến tiến trình con đang chạy ‘kill(pid, SIGINT);’.
 - Ngoài ra đoạn được tô vàng nhằm khai báo ‘pid’ được khai báo để lưu trữ process ID của tiến trình con. Điều này giúp chúng ta có thể gửi tín hiệu để kết thúc tiến trình con khi cần.

```
int main(void) {
    char *args[MAX_LINE];
    int should_run = 1;
    char input[MAX_LINE];
    char *args_pipe[MAX_LINE];

    while (should_run){
        signal(SIGINT, on_sigint);
        printf("it007sh>");
        fflush(stdout);

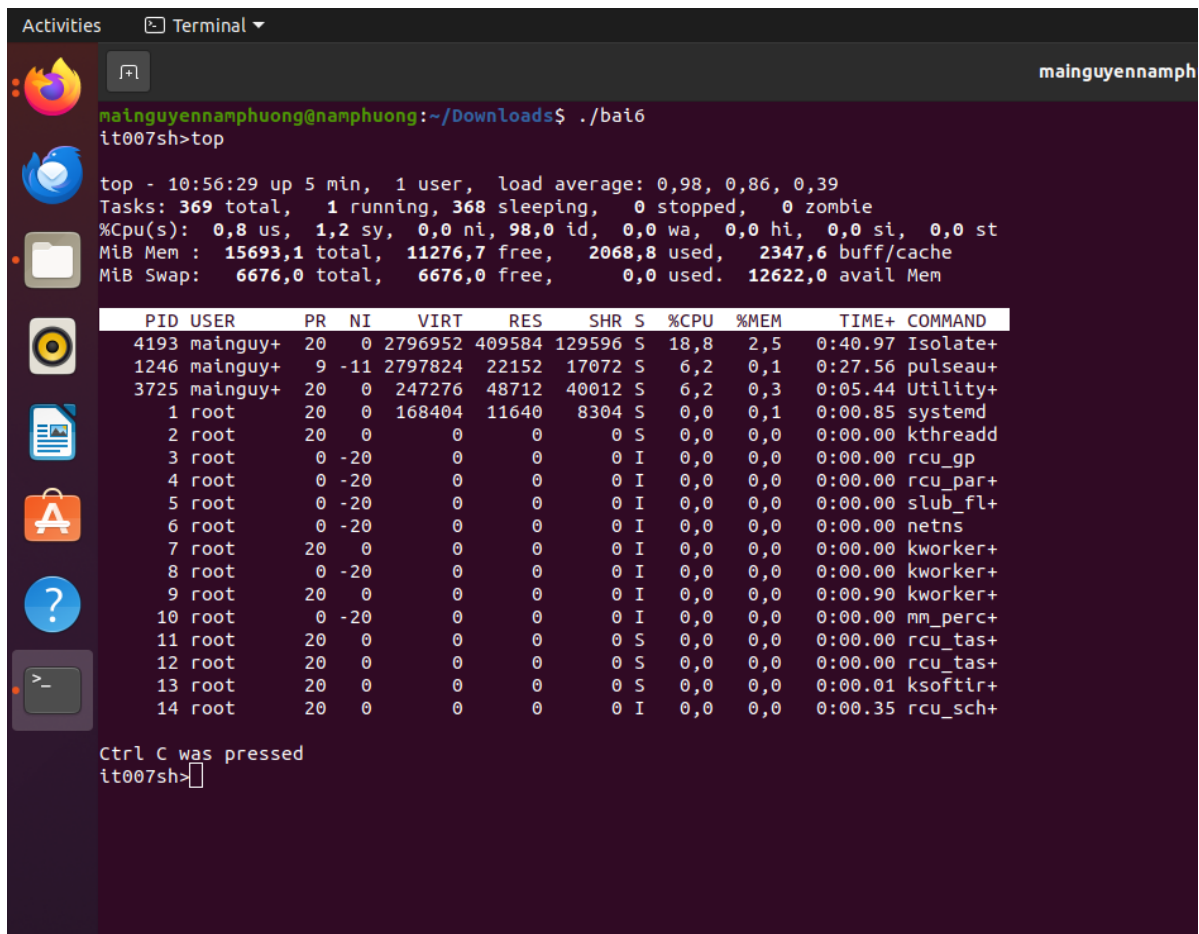
        fgets(input, MAX_LINE, stdin);
        input[strlen(input) - 1] = '\0';
    }
}
```

- Đoạn mã này đặt hàm on_sigint làm hàm xử lý cho tín hiệu SIGINT (Ctrl+C).

```
    }
    pid = fork();
    if (pid < 0){
        printf("Failed to fork.\n");
        exit(1);
    }
    else if (pid == 0){
        if (chuyenhuongdauvao){
            int fd = open(inputFile, O_RDONLY);
            dup2(fd, STDIN_FILENO);
            close(fd);
        }
        if (chuyenhuongdaura){
            int fd = open(outputFile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
            dup2(fd, STDOUT_FILENO);
            close(fd);
        }
        if (pipe_idx != -1){
            int fd[2];
            pipe(fd);
            pid_t pid2 = fork();
            if (pid2 == 0){
                close(fd[0]);
                dup2(fd[1], STDOUT_FILENO);
                close(fd[1]);
                execvp(args[0], args);
                exit(1);
            }
            else{
                close(fd[1]);
                dup2(fd[0], STDIN_FILENO);
                close(fd[0]);
                execvp(args_pipe[0], args_pipe);
                exit(1);
            }
        }
        else{
            execvp(args[0], args);
            perror("execvp failed");
            exit(1);
        }
    }
    else
        wait(NULL);
}
```

- Lệnh `pid = fork()` -> tạo một tiến trình con
- Nếu `pid = 0`, tức là đang ở trong tiến trình con, thực thi lệnh bằng `execvp`.
- Nếu `pid > 0`, tức là đang ở trong tiến trình cha, chờ tiến trình con kết thúc bằng `wait(NULL)`;
- ⇒ Vậy khi người dùng nhấn `Ctrl+C`:
- Tín hiệu `SIGINT` được gửi đến tiến trình cha.
- Hàm `on_sigint` được gọi và gửi tiếp tín hiệu `SIGINT` đến tiến trình con.
- Tiến trình con nhận tín hiệu và kết thúc.
- Tiến trình cha thoát khỏi hàm `wait(NULL)`; và hiển thị lại dấu nhắc `it007sh>`.

KẾT QUẢ:



```
mainguyennamphuong@namphuong:~/Downloads$ ./bai6
it007sh>top

top - 10:56:29 up 5 min, 1 user, load average: 0,98, 0,86, 0,39
Tasks: 369 total, 1 running, 368 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,8 us, 1,2 sy, 0,0 ni, 98,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
MiB Mem : 15693,1 total, 11276,7 free, 2068,8 used, 2347,6 buff/cache
MiB Swap: 6676,0 total, 6676,0 free, 0,0 used. 12622,0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
  4193 mainguy+  20   0 2796952 409584 129596 S   18,8   2,5   0:40.97 Isolate+
  1246 mainguy+  9  -11 2797824 22152  17072 S    6,2   0,1   0:27.56 pulseau+
  3725 mainguy+  20   0 247276  48712  40012 S    6,2   0,3   0:05.44 Utility+
    1 root      20   0 168404  11640  8304 S    0,0   0,1   0:00.85 systemd
    2 root      20   0      0      0      0 S    0,0   0,0   0:00.00 kthreadd
    3 root       0 -20      0      0      0 I    0,0   0,0   0:00.00 rcu_gp
    4 root       0 -20      0      0      0 I    0,0   0,0   0:00.00 rcu_par+
    5 root       0 -20      0      0      0 I    0,0   0,0   0:00.00 slub_fl+
    6 root       0 -20      0      0      0 I    0,0   0,0   0:00.00 netns
    7 root      20   0      0      0      0 I    0,0   0,0   0:00.00 kworker+
    8 root       0 -20      0      0      0 I    0,0   0,0   0:00.00 kworker+
    9 root      20   0      0      0      0 I    0,0   0,0   0:00.90 kworker+
   10 root       0 -20      0      0      0 I    0,0   0,0   0:00.00 mm_perc+
   11 root      20   0      0      0      0 S    0,0   0,0   0:00.00 rcu_tas+
   12 root      20   0      0      0      0 S    0,0   0,0   0:00.00 rcu_tas+
   13 root      20   0      0      0      0 S    0,0   0,0   0:00.01 ksofttir+
   14 root      20   0      0      0      0 I    0,0   0,0   0:00.35 rcu_sch+

Ctrl C was pressed
it007sh>
```