Name: Nguyễn Khoa Nguyên

ID: 24521190

Class: IT007.Q15.1

OPERATING SYSTEM
LAB 6'S REPORT

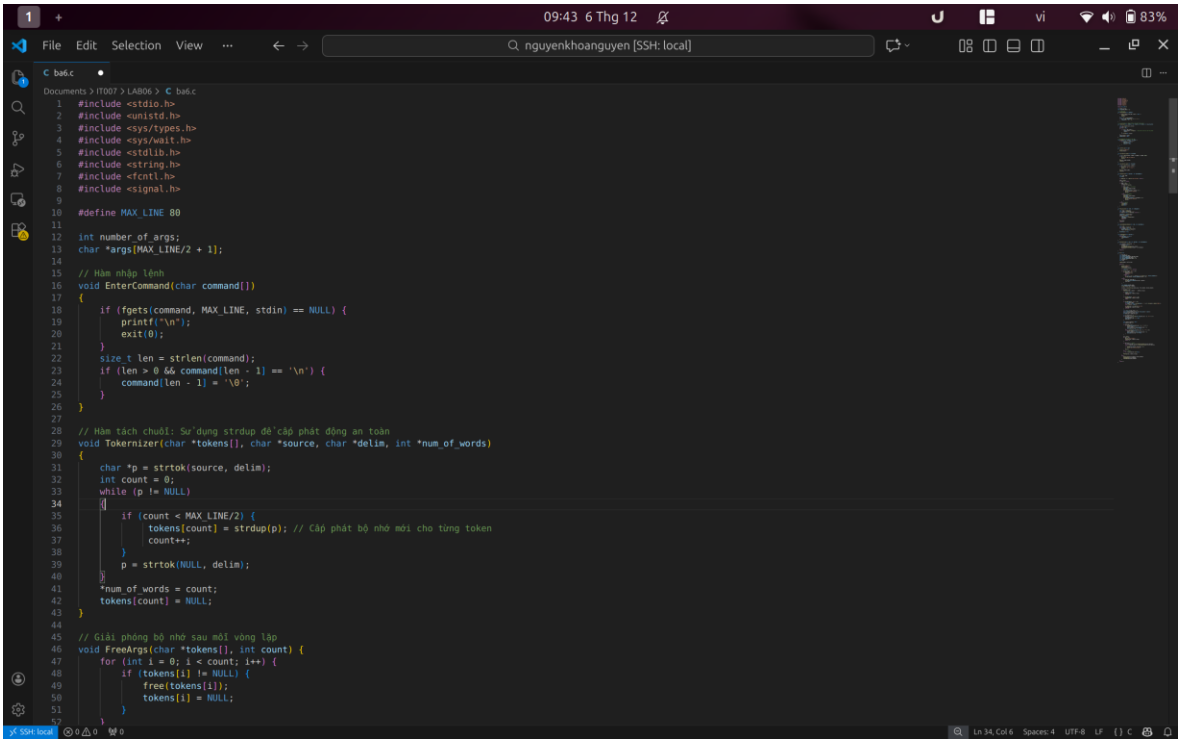
SUMMARY

Self-scrores:

Note: Export file to **PDF and name the file by following format:*
Student ID_LABx.pdf

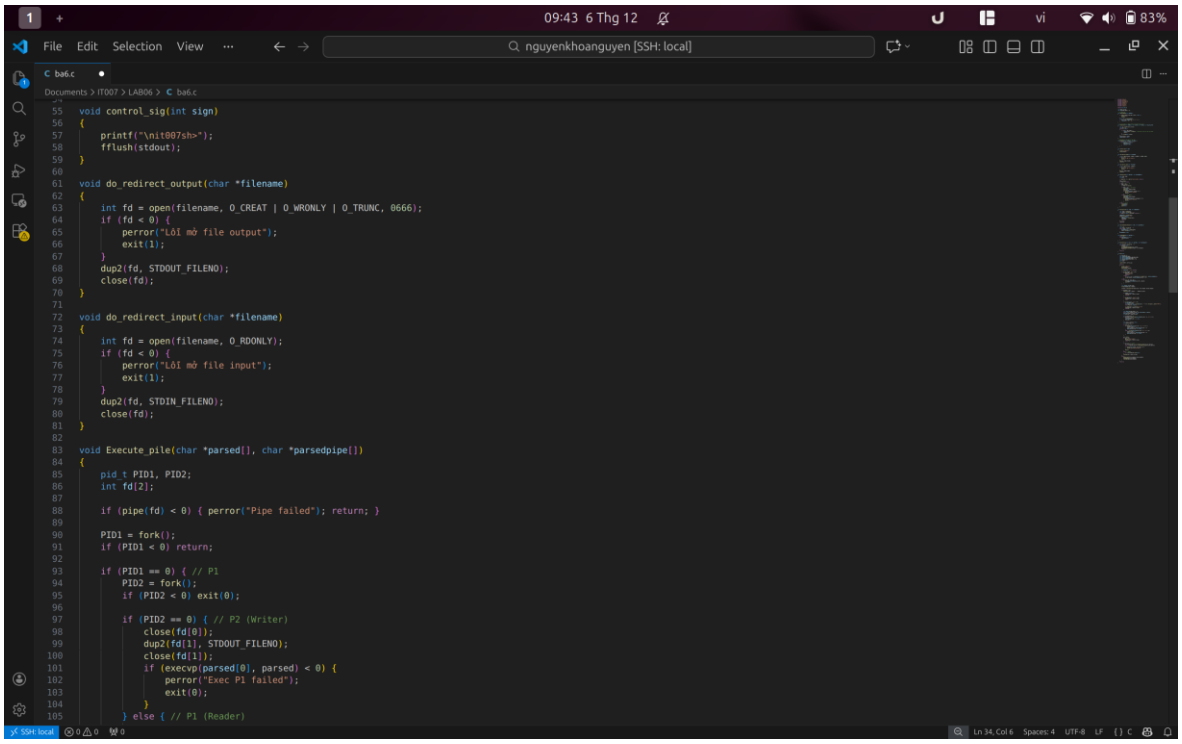
Section 6.4

1. Source code



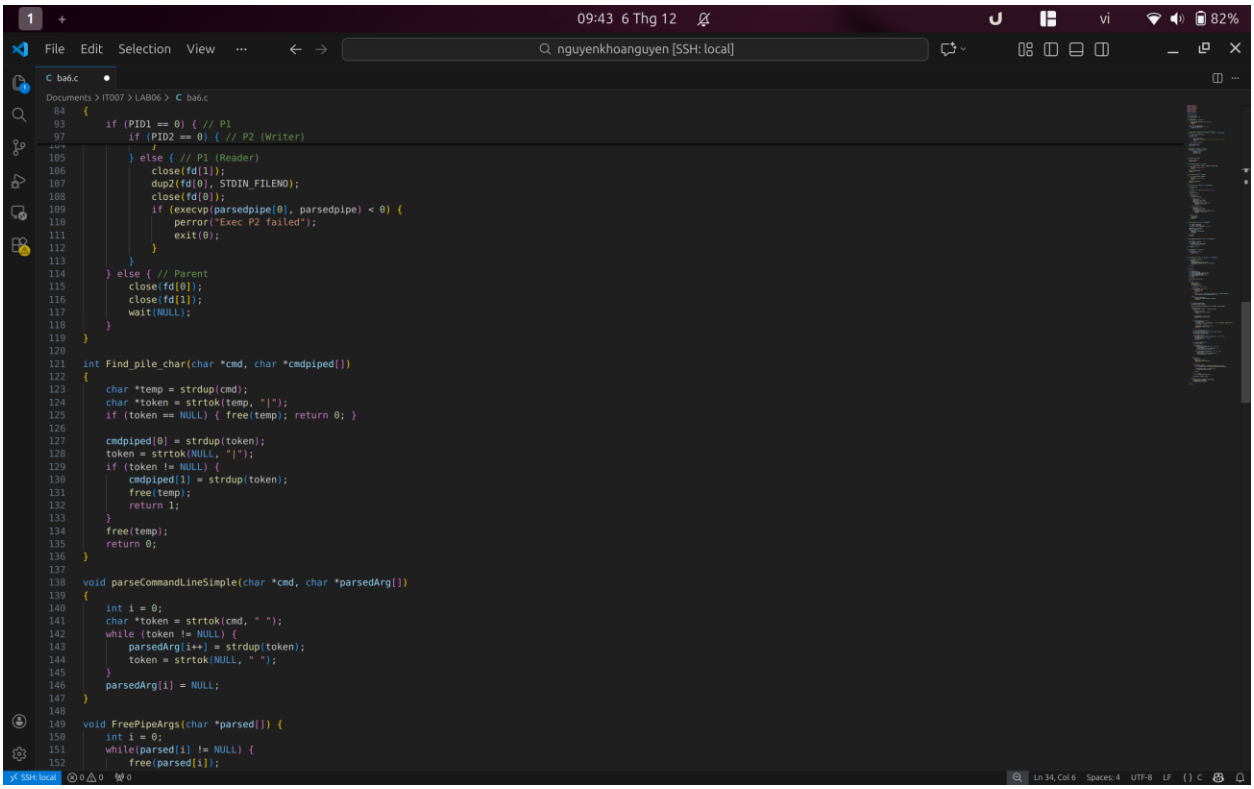
```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>

#define MAX_LINE 80

int number_of_args;
char *args[MAX_LINE/2 + 1];

// Hàm nhập lệnh
void EnterCommand(char command[])
{
    if (fgets(command, MAX_LINE, stdin) == NULL) {
        printf("\n");
        exit(0);
    }
    size_t len = strlen(command);
    if (len > 0 && command[len - 1] == '\n') {
        command[len - 1] = '\0';
    }
}

// Hàm tách chuỗi: Sử dụng strdup để cấp phát động an toàn
void Tokernizer(char *tokens[], char *source, char *delim, int *num_of_words)
{
    char *p = strtok(source, delim);
    int count = 0;
    while (p != NULL)
    {
        if (count < MAX_LINE/2) {
            tokens[count] = strdup(p); // Cấp phát bộ nhớ mới cho từng token
            count++;
        }
        p = strtok(NULL, delim);
    }
    *num_of_words = count;
    tokens[count] = NULL;
}

// Giải phóng bộ nhớ sau mỗi vòng lặp
void FreeArgs(char *tokens[], int count) {
    for (int i = 0; i < count; i++) {
        if (tokens[i] != NULL) {
            free(tokens[i]);
            tokens[i] = NULL;
        }
    }
```

Hình 1 Nội dung code (1)



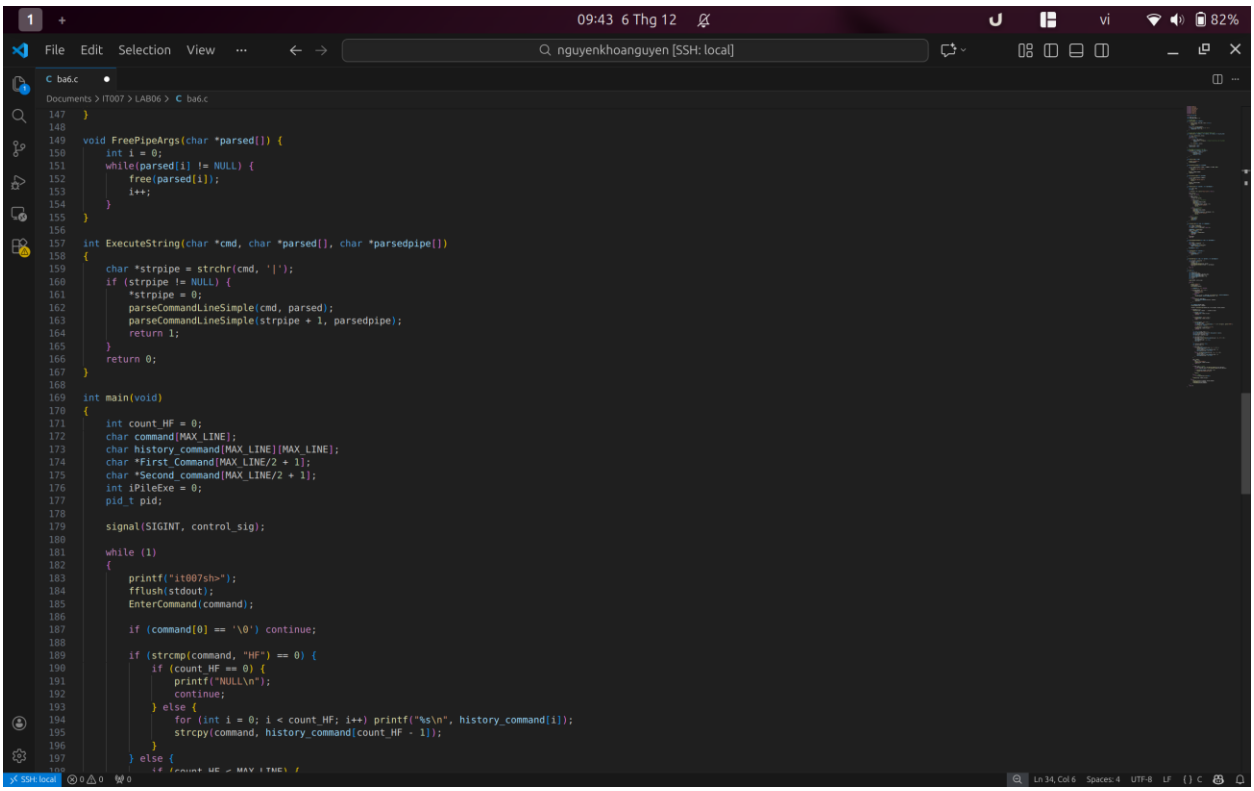
```c
void control_sig(int sign)
{
    printf("\nit007sh>");
    fflush(stdout);
}

void do_redirect_output(char *filename)
{
    int fd = open(filename, O_CREAT | O_WRONLY | O_TRUNC, 0666);
    if (fd < 0) {
        perror("Lỗi mở file output");
        exit(1);
    }
    dup2(fd, STDOUT_FILENO);
    close(fd);
}

void do_redirect_input(char *filename)
{
    int fd = open(filename, O_RDONLY);
    if (fd < 0) {
        perror("Lỗi mở file input");
        exit(1);
    }
    dup2(fd, STDIN_FILENO);
    close(fd);
}

void Execute_pile(char *parsed[], char *parsedpipe[])
{
    pid_t PID1, PID2;
    int fd[2];

    if (pipe(fd) < 0) { perror("Pipe failed"); return; }

    PID1 = fork();
    if (PID1 < 0) return;

    if (PID1 == 0) { // P1
        PID2 = fork();
        if (PID2 < 0) exit(0);

        if (PID2 == 0) { // P2 (Writer)
            close(fd[0]);
            dup2(fd[1], STDOUT_FILENO);
            close(fd[1]);
            if (execvp(parsed[0], parsed) < 0) {
                perror("Exec P1 failed");
                exit(0);
            }
        } else { // P1 (Reader)
```

Hình 2 Nội dung code (2)



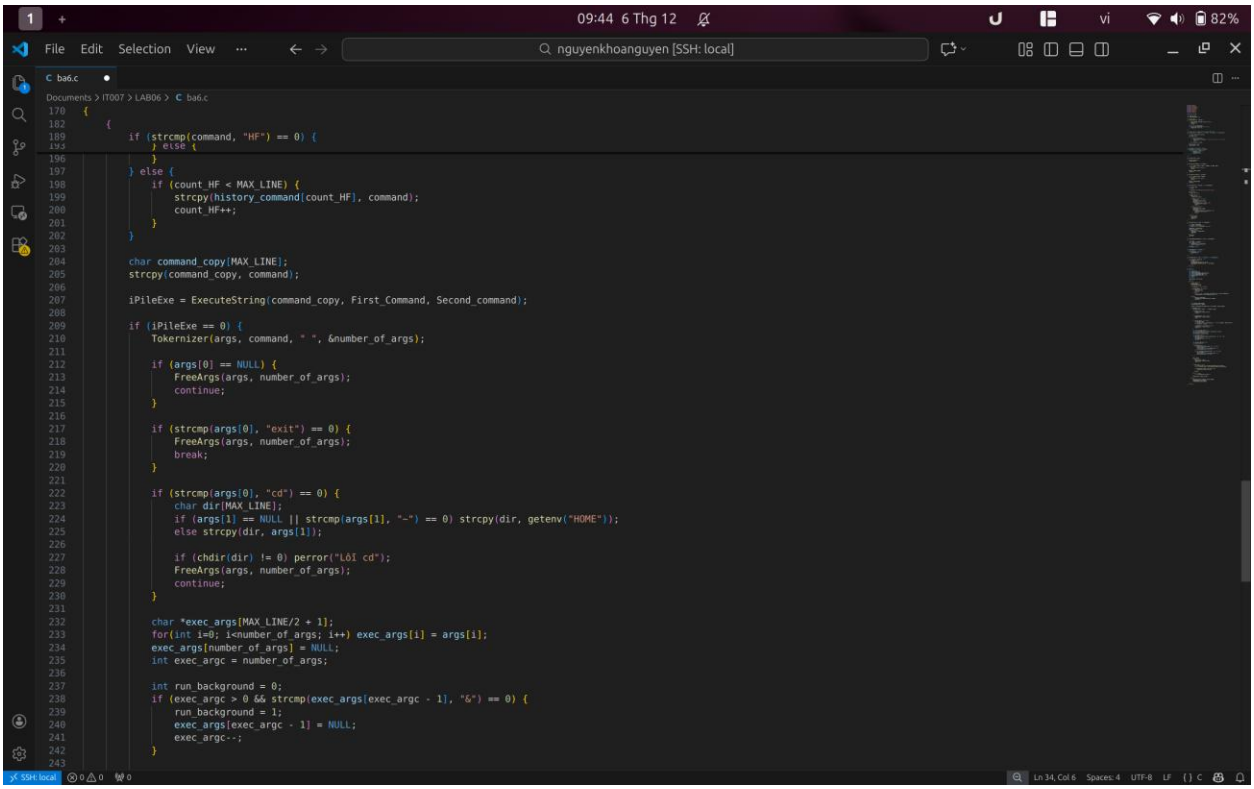
```c
{
    if (PID1 == 0) { // P1
        if (PID2 == 0) { // P2 (Writer)
            /
        } else { // P1 (Reader)
            close(fd[1]);
            dup2(fd[0], STDIN_FILENO);
            close(fd[0]);
            if (execvp(parsedpipe[0], parsedpipe) < 0) {
                perror("Exec P2 failed");
                exit(0);
            }
        }
    } else { // Parent
        close(fd[0]);
        close(fd[1]);
        wait(NULL);
    }
}

int Find_pile_char(char *cmd, char *cmdpiped[])
{
    char *temp = strdup(cmd);
    char *token = strtok(temp, "|");
    if (token == NULL) { free(temp); return 0; }

    cmdpiped[0] = strdup(token);
    token = strtok(NULL, "|");
    if (token != NULL) {
        cmdpiped[1] = strdup(token);
        free(temp);
        return 1;
    }
    free(temp);
    return 0;
}

void parseCommandLineSimple(char *cmd, char *parsedArg[])
{
    int i = 0;
    char *token = strtok(cmd, " ");
    while (token != NULL) {
        parsedArg[i++] = strdup(token);
        token = strtok(NULL, " ");
    }
    parsedArg[i] = NULL;
}

void FreePipeArgs(char *parsed[]) {
    int i = 0;
    while(parsed[i] != NULL) {
        free(parsed[i]);
```

Hình 3 Nội dung code (3)



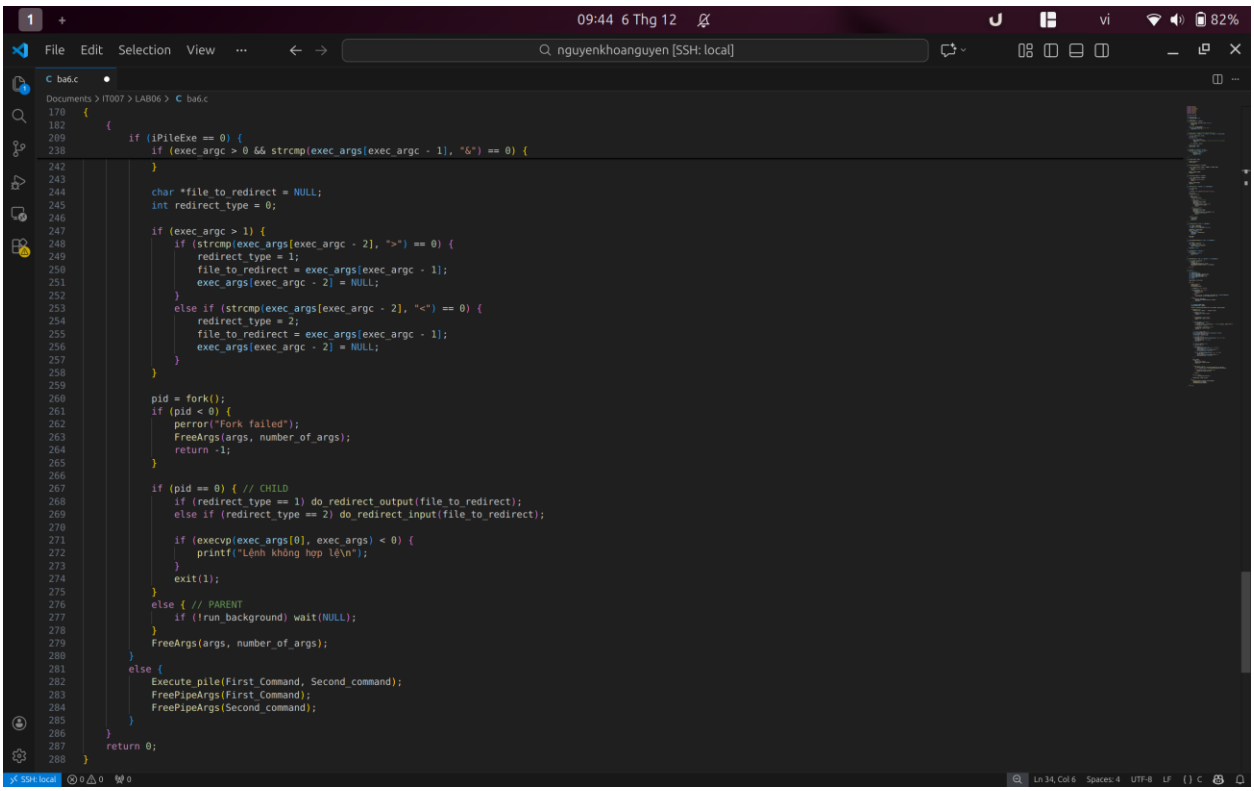
```c
}

void FreePipeArgs(char *parsed[]) {
    int i = 0;
    while(parsed[i] != NULL) {
        free(parsed[i]);
        i++;
    }
}

int ExecuteString(char *cmd, char *parsed[], char *parsedpipe[])
{
    char *strpipe = strchr(cmd, '|');
    if (strpipe != NULL) {
        *strpipe = 0;
        parseCommandLineSimple(cmd, parsed);
        parseCommandLineSimple(strpipe + 1, parsedpipe);
        return 1;
    }
    return 0;
}

int main(void)
{
    int count_HF = 0;
    char command[MAX_LINE];
    char history_command[MAX_LINE][MAX_LINE];
    char *First_Command[MAX_LINE/2 + 1];
    char *Second_command[MAX_LINE/2 + 1];
    int iPileExe = 0;
    pid_t pid;

    signal(SIGINT, control_sig);

    while (1)
    {
        printf("it007sh>");
        fflush(stdout);
        EnterCommand(command);

        if (command[0] == '\0') continue;

        if (strcmp(command, "HF") == 0) {
            if (count_HF == 0) {
                printf("NULL\n");
                continue;
            } else {
                for (int i = 0; i < count_HF; i++) printf("%s\n", history_command[i]);
                strcpy(command, history_command[count_HF - 1]);
            }
        } else {
```

Hình 4 Nội dung code (4)



```
{
    {
        if (strcmp(command, "HF") == 0) {
            } else {
            }
        } else {
            if (count_HF < MAX_LINE) {
                strcpy(history_command[count_HF], command);
                count_HF++;
            }
        }

        char command_copy[MAX_LINE];
        strcpy(command_copy, command);

        iPileExe = ExecuteString(command_copy, First_Command, Second_command);

        if (iPileExe == 0) {
            Tokernizer(args, command, " ", &number_of_args);

            if (args[0] == NULL) {
                FreeArgs(args, number_of_args);
                continue;
            }

            if (strcmp(args[0], "exit") == 0) {
                FreeArgs(args, number_of_args);
                break;
            }

            if (strcmp(args[0], "cd") == 0) {
                char dir[MAX_LINE];
                if (args[1] == NULL || strcmp(args[1], "~") == 0) strcpy(dir, getenv("HOME"));
                else strcpy(dir, args[1]);

                if (chdir(dir) != 0) perror("Lỗi cd");
                FreeArgs(args, number_of_args);
                continue;
            }

            char *exec_args[MAX_LINE/2 + 1];
            for(int i=0; i<number_of_args; i++) exec_args[i] = args[i];
            exec_args[number_of_args] = NULL;
            int exec_argc = number_of_args;

            int run_background = 0;
            if (exec_argc > 0 && strcmp(exec_args[exec_argc - 1], "&") == 0) {
                run_background = 1;
                exec_args[exec_argc - 1] = NULL;
                exec_argc--;
            }
```

Hình 5 Nội dung code (5)



```
{
    {
        if (iPileExe == 0) {
            if (exec_argc > 0 && strcmp(exec_args[exec_argc - 1], "&") == 0) {
            }

            char *file_to_redirect = NULL;
            int redirect_type = 0;

            if (exec_argc > 1) {
                if (strcmp(exec_args[exec_argc - 2], ">") == 0) {
                    redirect_type = 1;
                    file_to_redirect = exec_args[exec_argc - 1];
                    exec_args[exec_argc - 2] = NULL;
                }
                else if (strcmp(exec_args[exec_argc - 2], "<") == 0) {
                    redirect_type = 2;
                    file_to_redirect = exec_args[exec_argc - 1];
                    exec_args[exec_argc - 2] = NULL;
                }
            }

            pid = fork();
            if (pid < 0) {
                perror("Fork failed");
                FreeArgs(args, number_of_args);
                return -1;
            }

            if (pid == 0) { // CHILD
                if (redirect_type == 1) do_redirect_output(file_to_redirect);
                else if (redirect_type == 2) do_redirect_input(file_to_redirect);

                if (execvp(exec_args[0], exec_args) < 0) {
                    printf("Lệnh không hợp lệ\n");
                }
                exit(1);
            }
            else { // PARENT
                if (!run_background) wait(NULL);
            }
            FreeArgs(args, number_of_args);
        }
        else {
            Execute_pile(First_Command, Second_command);
            FreePipeArgs(First_Command);
            FreePipeArgs(Second_command);
        }
    }
    return 0;
}
```

Hình 6 Nội dung code (6)

- Giải thích code

- Các thư viện và hằng số

- Các thư viện <stdio.h>, <unistd.h>, <syswait.h>,... cung cấp các hàm hệ thống để thao tác nhập/xuất, tạo tiến trình, thực thi lệnh và xử lý các tín hiệu
- #define MAX_LINE 80: Quy định độ dài tối đa của một câu lệnh là 80 ký tự

- Các hàm hỗ trợ xử lý chuỗi và bộ nhớ

- EnterCommand(char command[]): Dùng để đọc dòng lệnh người dùng nhập từ bàn phím. Thực hiện bằng cách dùng fgets để đọc và xóa các ký tự xuống dòng ở cuỗi chuỗi để các lệnh hệ thống hiểu được
- Tokernizer: Dùng để tách chuỗi lệnh dài thành các từ riêng biệt. Thực hiện bằng cách sử dụng strdup(p) để cấp phát bộ nhớ động riêng biệt cho từng từ
- FreeArgs() và FreePipeArgs(): Dùng để dọn dẹp bộ nhớ sau khi thực thi xong một lệnh vì Tokernizer dùng strdup để cấp phát động, nếu không giải phóng thì máy có thể bị memory leak sau một thời gian chạy

- Xử lý tín hiệu và I/O

- control_sig(int sign): Dùng để xử lý tín hiệu ngắt, ví dụ khi ấn Ctrl + C thay vì chương trình bị tắt ngầm thì hàm này sẽ chặn tín hiệu đó và chỉ in ra dòng nhắc it007sh> mới
- do_redirect_output(char *filename): Dùng để xử lý toán từ >. Thực hiện bằng cách mở file và dùng dup2 để thay thế STDOUT bằng file đó. Kết quả lệnh sẽ chui vào file thay vì hiện lên màn hình
- do_redirect_input(char *filename): Dùng để xử lý toán tử <. Thực hiện bằng cách dùng dup2 để thay thế STDIN bằng nội dung file và lệnh sẽ đọc dữ liệu từ file

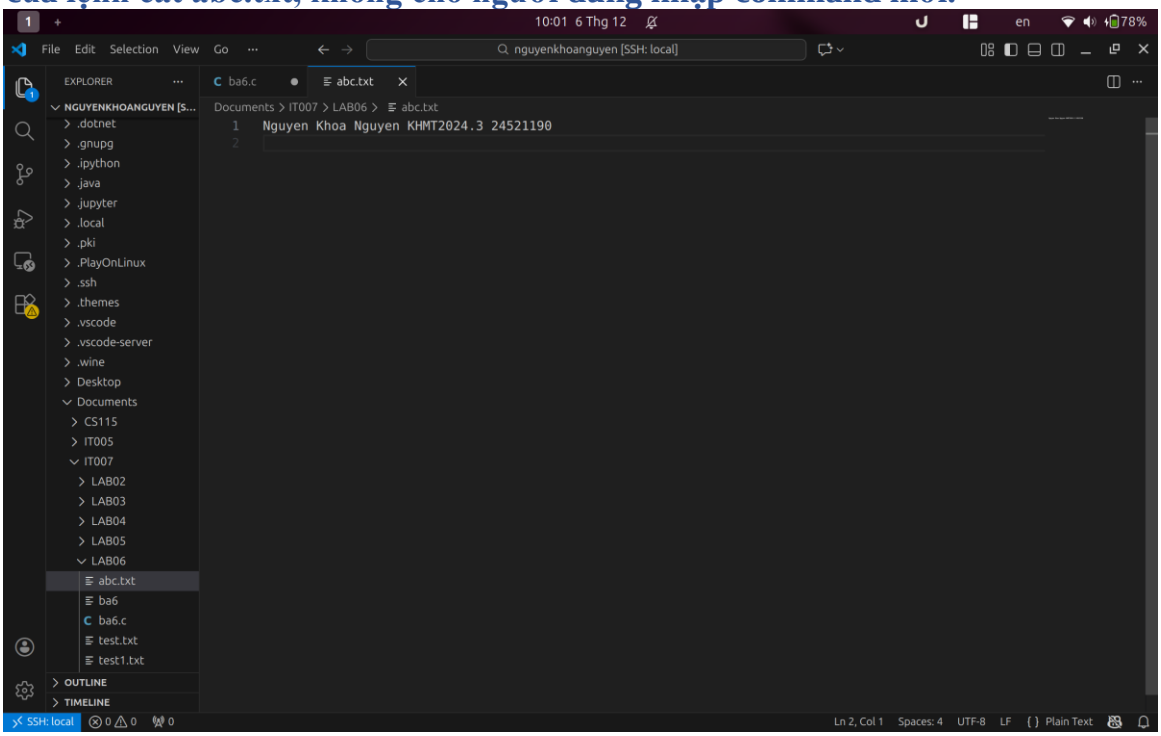
- Xử lý Pipe

- Execute_pile(): Dùng để thực thi hai lệnh nối nhau bởi dấu |. Thực hiện bằng cách tạo một pipe, sau đó fork để tạo tiến trình con, cháu. P2(Writer) để chạy lệnh đầu tiên, chuyển đầu ra vào đầu ghi của ống. P1(Reader) chạy lệnh thứ hai, chuyển đầu vào lấy từ đầu đọc của ống. Parent(shell) dùng để đóng toàn bộ ống và chờ con xong
- Find_pile_char, parseCommandLineSimple, ExecuteString: Nhóm hàm này phát hiện dấu |, cắt đôi câu lệnh thành 2 phần sau đó tách từ cho từng phần để chuẩn bị đưa vào Execute_pile

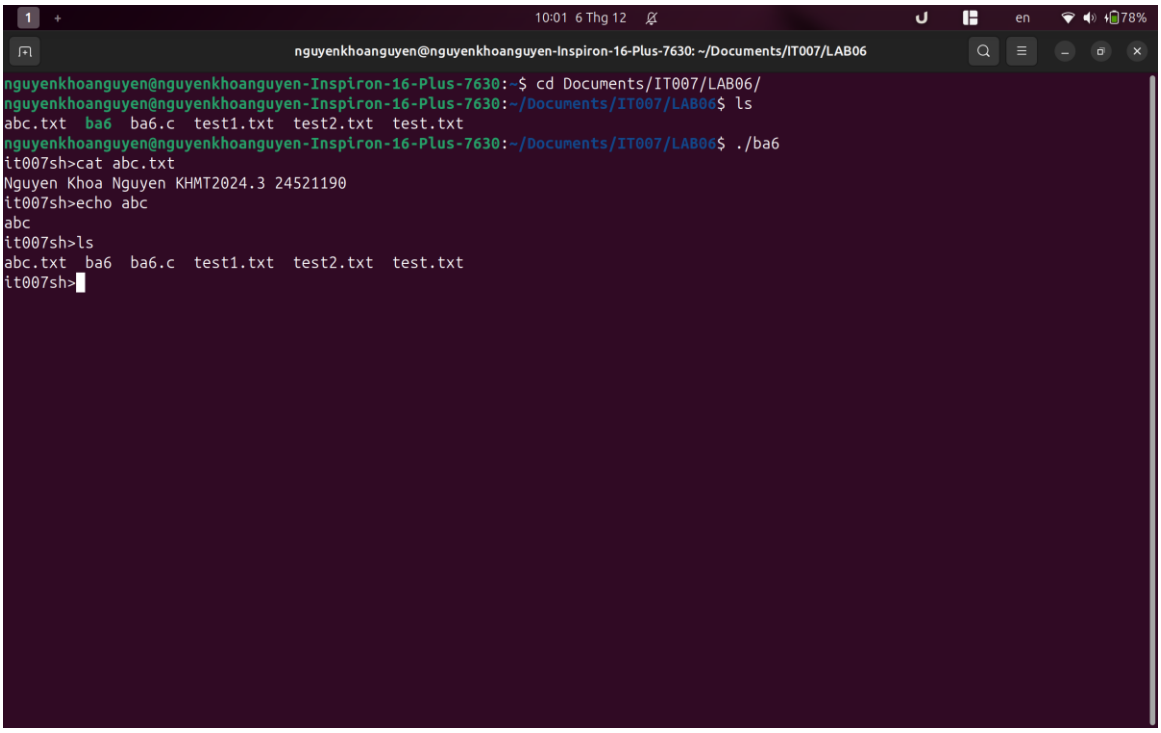
- Hàm main

- Một vòng lặp vô hạn điều khiển luồng chạy của Shell
- Nếu là lệnh pipe thì gọi ExecuteString() để xem có dấu | không, nếu có thì chạy theo luồng Pipe
- Nếu không phải là lệnh pipe: Tách chuỗi lệnh (Tokernizer), và thực thi các hàm được người dùng nhập

Câu 1: Thực thi command trong tiến trình con. Ví dụ: khi thực hiện it007sh> cat abc.txt. Kết quả sẽ hiển thị nội dung của file abc.txt, kết thúc dòng lệnh sẽ hiển thị dấu nhắc it007sh> để người dùng nhập lệnh tiếp theo. Lưu ý rằng trong thời thực thi của lệnh cat abc.txt, không cho người dùng nhập command mới.

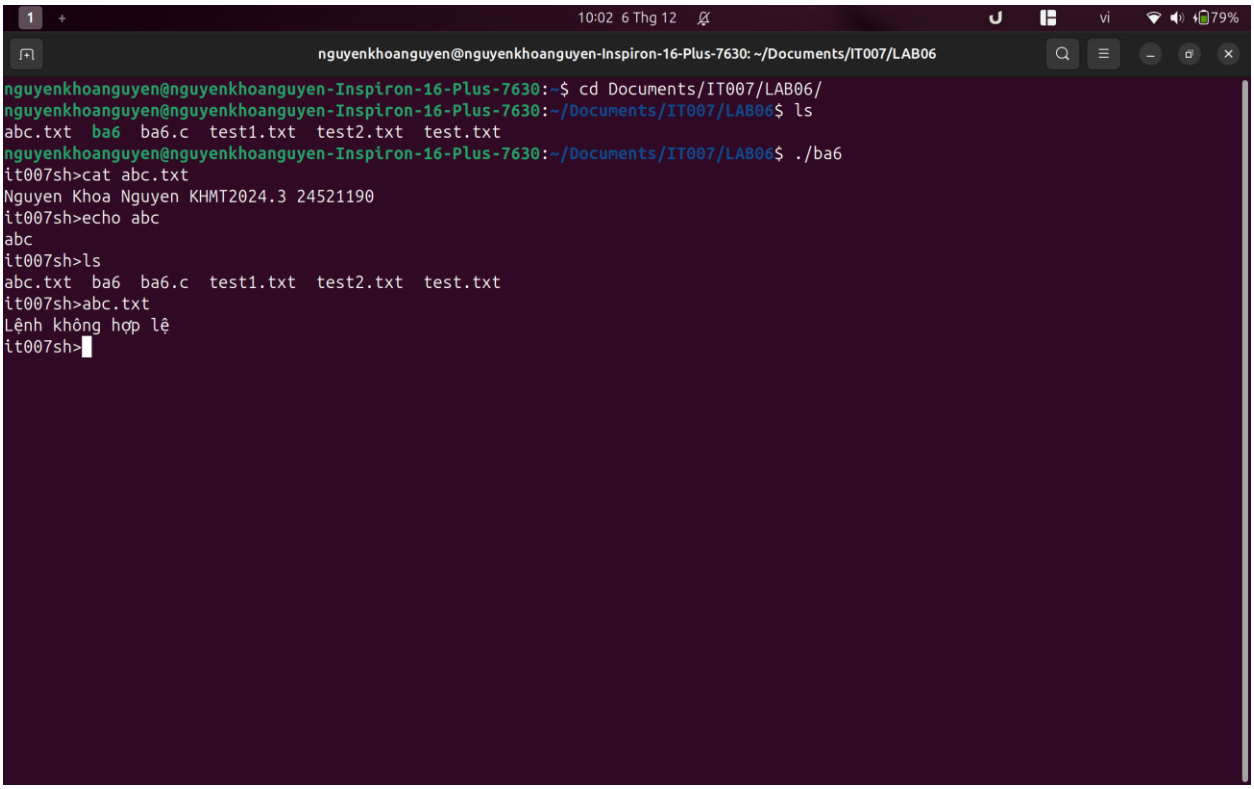


Hình 7 Nội dung file abc.txt

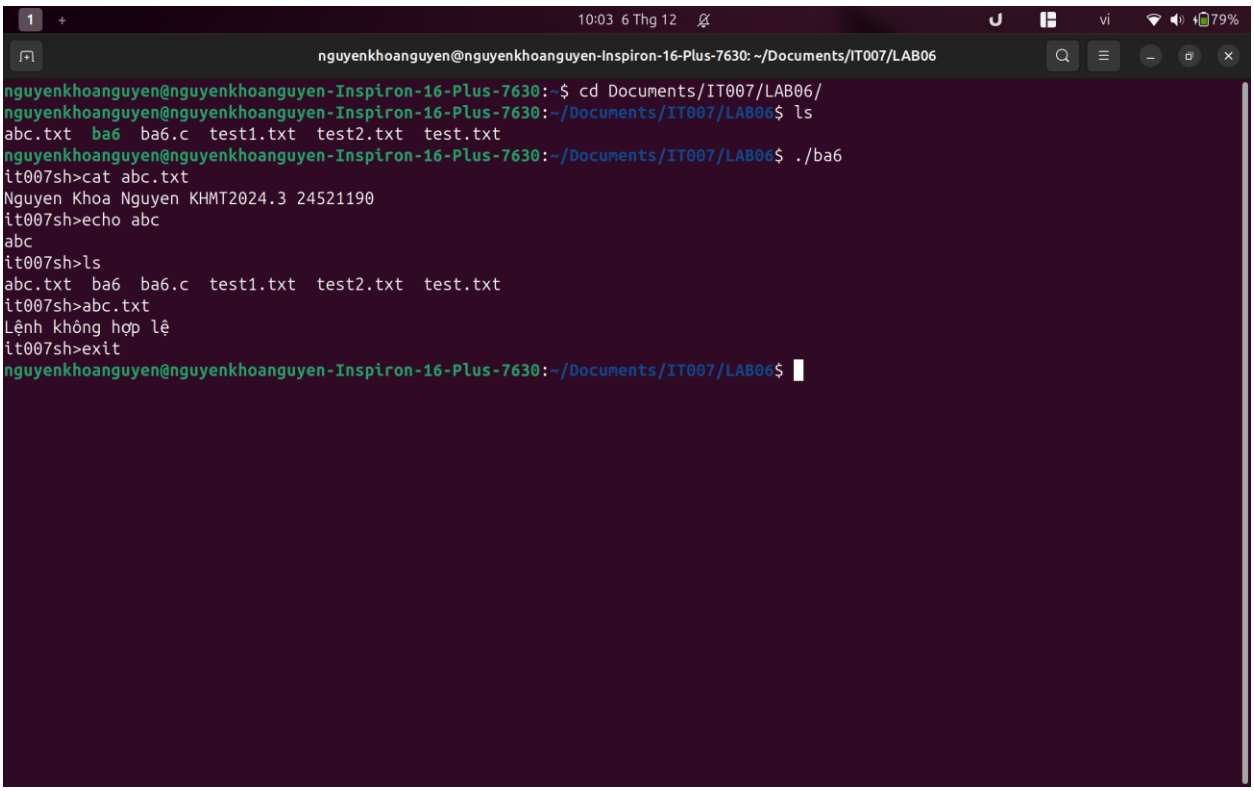


Hình 8 Kết quả khi thực thi các lệnh cơ bản

- Khi gõ sai lệnh



Hình 9 Kết quả nếu chúng ta nhập sai lệnh

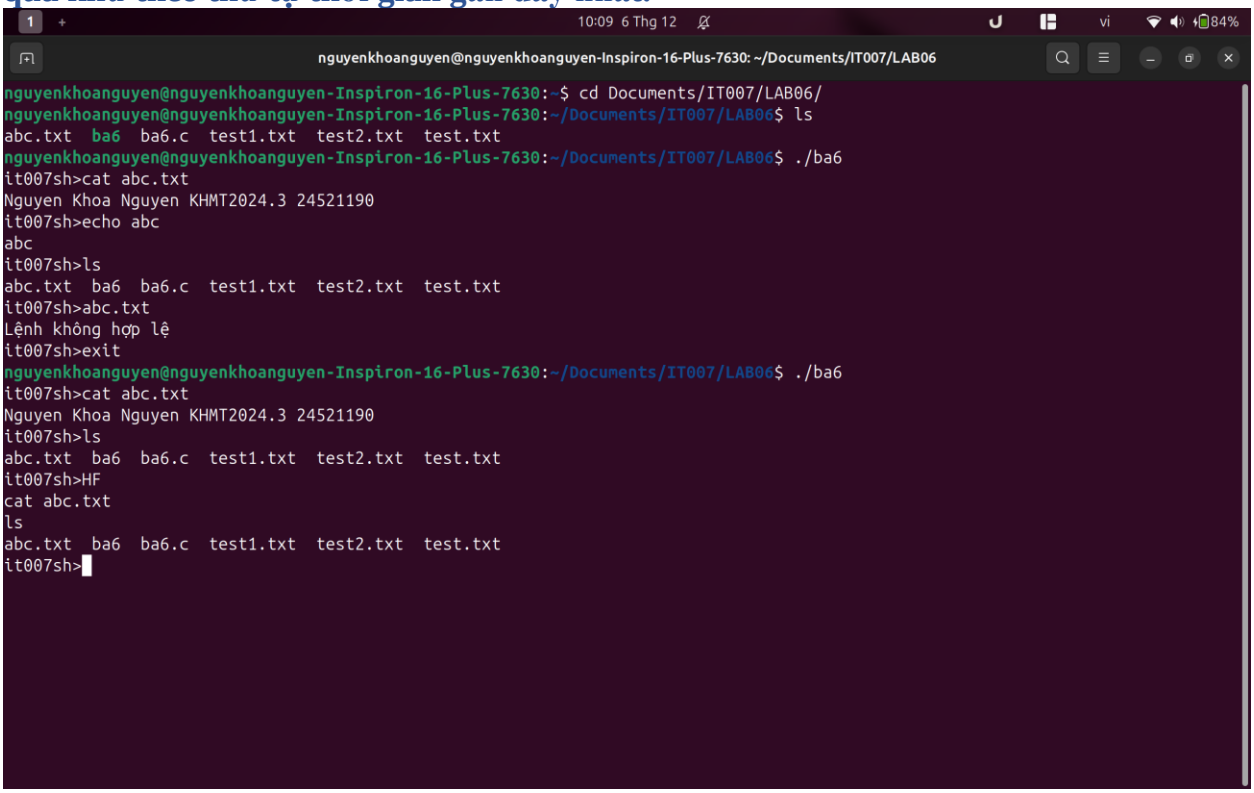


Hình 10 Nhập lệnh exit để thoát chương trình

- Giải thích kết quả:

- Hàm EnterCommand đọc chuỗi "cat abc.txt"
- Tokernizer: Tách thành mảng args: ["cat", "abc.txt", NULL]
- Nếu không có | (pipe), không có redirect (>, <), không phải exit hay cd
- Tạo tiến trình con bằng fork
- Thực thi execv: Tiến trình con biến đổi thành chương trình cat, đọc file abc.txt và in nội dung "Nguyen Khoa Nguyen KHMT2024.3 24521190" ra màn hình

Câu 2: Tạo tính năng sử dụng lại câu lệnh gần đây nhất. Cung cấp History Feature, cho phép người dùng thực thi lệnh gần đây nhất bằng cách nhập 'HF' và nhấn enter. Khi sử dụng ký tự 'HF', shell sẽ lần lượt hiện lại các lệnh đã sử dụng trong quá khứ theo thứ tự thời gian gần đây nhất.



Hình 11 Kết quả khi ta nhập lệnh HF

- Giải thích kết quả:

- Trước khi gõ HF, chương trình đã chạy qua một vài lệnh: cat abc.txt và ls. Do đó mảng history_command đang chứa ["cat abc.txt", "ls"] và biến count_HF có giá trị là 2
- Khi nhập lệnh HF, chương trình so sánh chuỗi vừa nhập: strcmp(command, "HF") == 0. Kết quả là đúng nên nhảy vào xử lý lịch sử

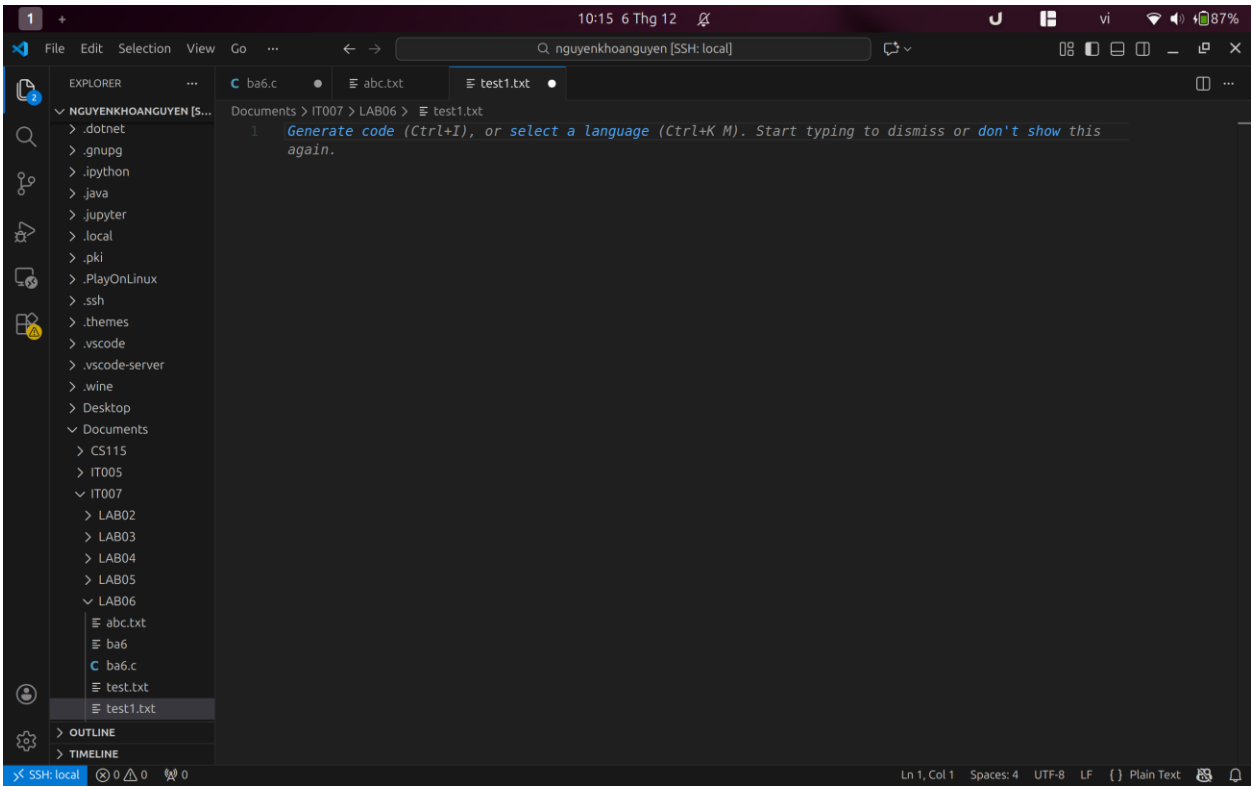
- strcpy(command, history_command[count_HF - 1]): Lấy lệnh tại vị trí cuối cùng và copy đè lên biến command, do đó bây giờ command không còn là "HF" nữa mà chính là "ls"

Câu 3: Chuyển hướng vào ra Hỗ trợ các toán tử chuyển hướng '>' và '<', trong đó '>' chuyển hướng đầu ra của lệnh sang một tệp và '<' chuyển hướng đầu vào của lệnh từ một tệp.

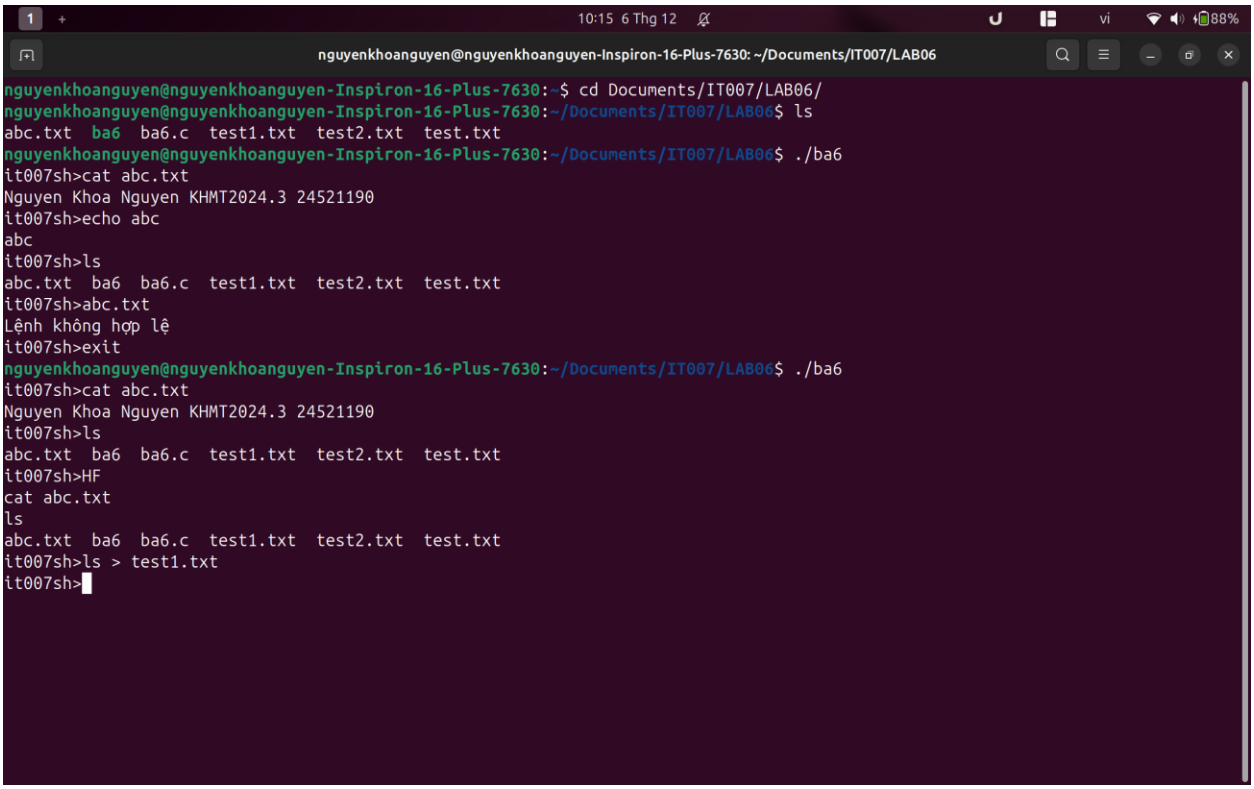
- Ví dụ: nếu người dùng nhập it007sh>ls > out.txt. Đầu ra từ lệnh ls sẽ được chuyển hướng đến tệp out.txt. Tương tự, đầu vào cũng có thể được chuyển hướng.

- Ví dụ, nếu người dùng nhập it007sh>sort < in.txt. Tệp in.txt sẽ đóng vai trò là đầu vào cho lệnh sắp xếp.

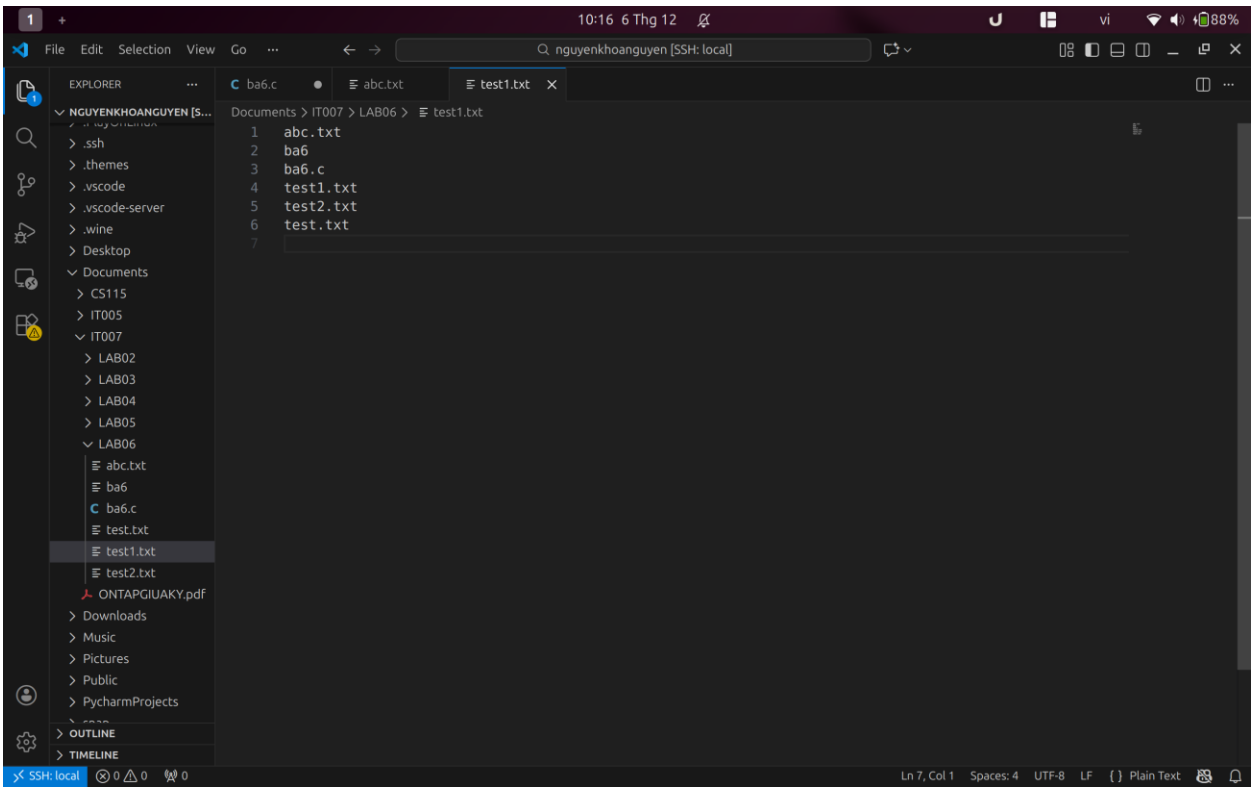
- Testcase 1: Thực thi lệnh ls > test1.txt



Hình 12 Nội dung ban đầu của file test1.txt



Hình 13 Nhập lệnh ls > test1.txt

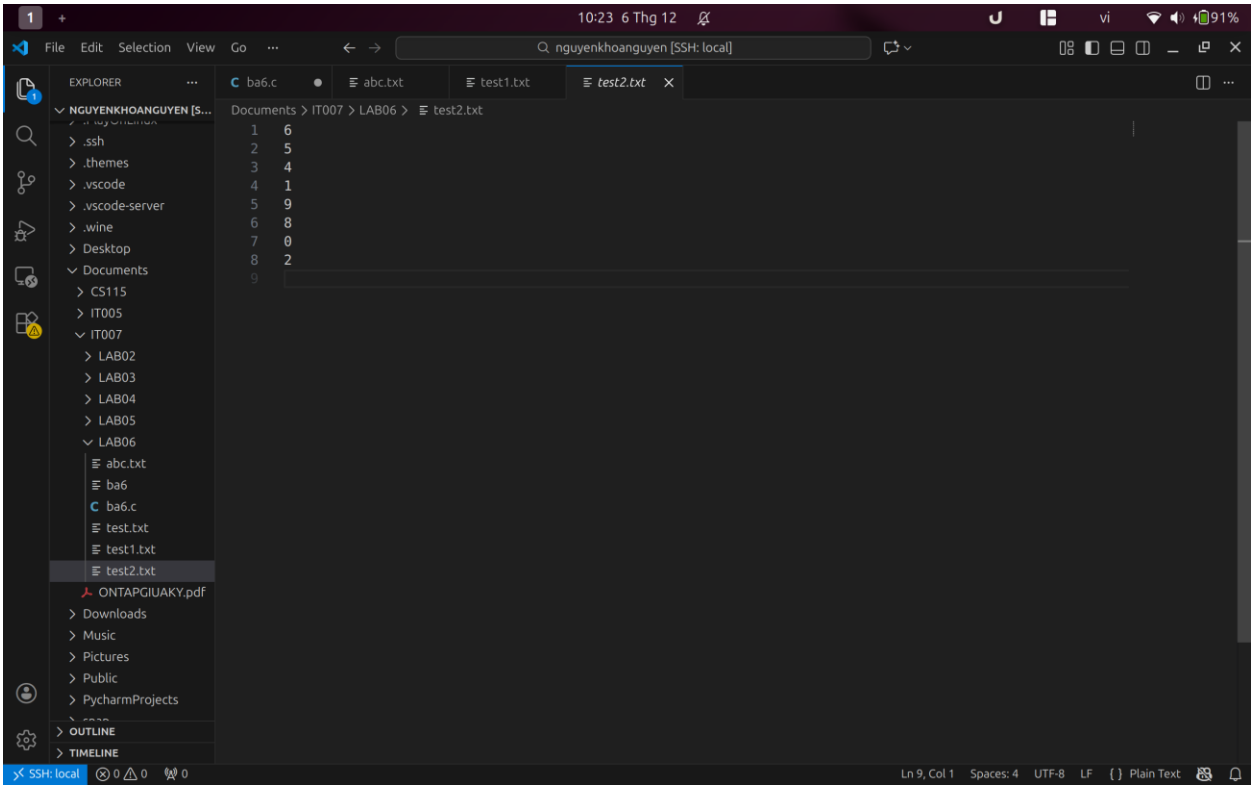


Hình 14 Nội dung của file test1.txt sau khi thực hiện nhập lệnh ls

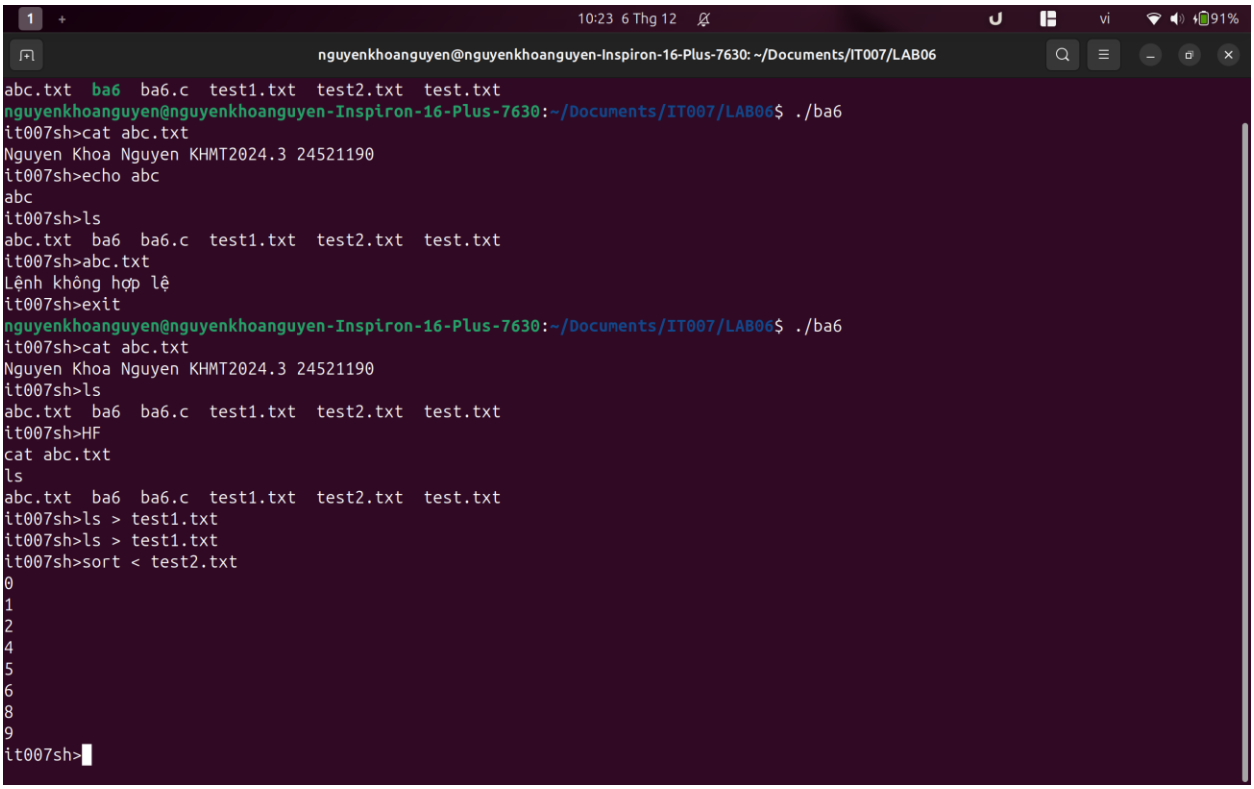
- Giải thích kết quả:

- Khi nhập lệnh ls > test1.txt thì mảng exec_args ban đầu là ["ls", ">", "test1.txt", NULL]
- Chương trình quét thấy kí tự ">" và xác định đây là output redirect và file đích chính là "test1.txt". Chương trình gán NULL vào vị trí dấu ">" và mảng exec_args chỉ còn ["ls", NULL]
- open("test1.txt"): Cờ O_TRUNC sẽ xóa sạch nội dung trong file nếu nó đã tồn tại
- dup2(fd, STDOUT_FILENO): FD 1 bây giờ không trỏ vào màn hình nữa mà trỏ thẳng vào file test1.txt, do đó màn hình không hiện gì cả mà chỉ có nội dung trong file

- Testcase 2: Thực thi lệnh sort < test2.txt



Hình 15 Nội dung ban đầu của file test2.txt



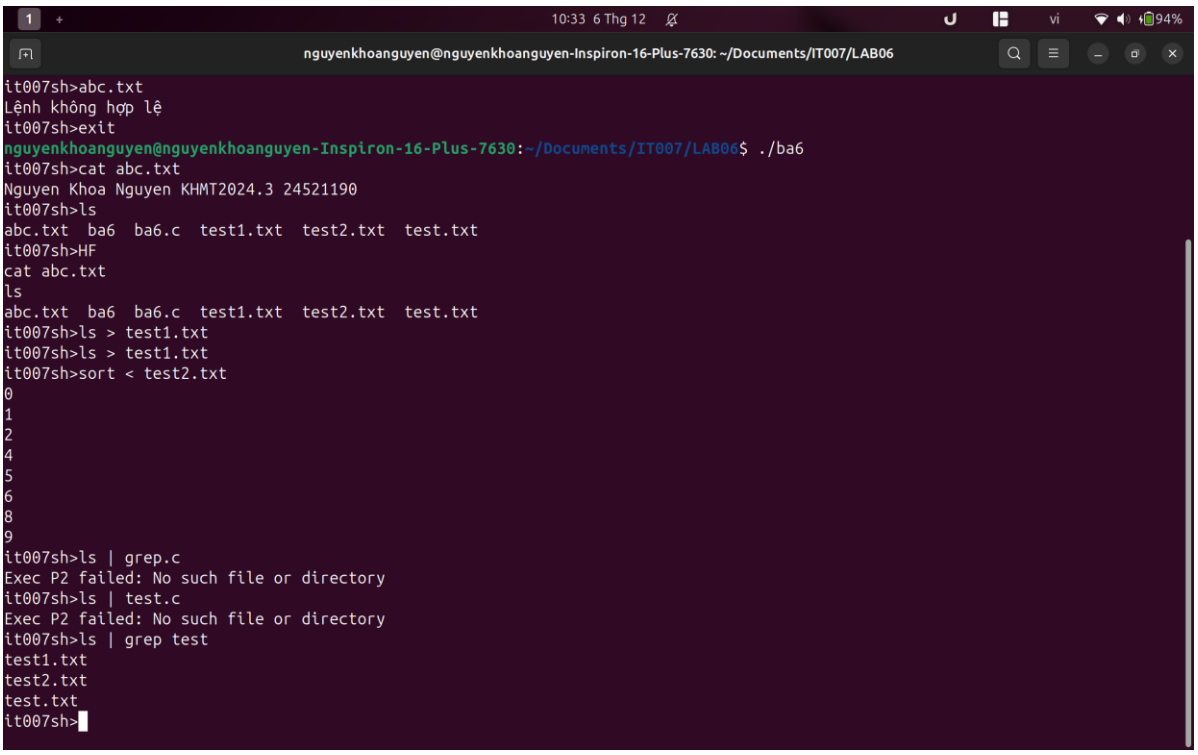
Hình 16 Thực thi lệnh sort < test2.txt

- Giải thích kết quả:

- Vì quét thấy "<" nên đặt redirect_type = 2, lưu tên file cần đọc: file_to_redirect = "test2.txt". Sau đó cắt lệnh và gán NULL vào vị trí dấu "<"
- fd = open("test2.txt", O_RDONLY): Mở file ở chế độ chỉ đọc
- dup2(fd, STDIN_FILENO): Khi bất kỳ lệnh nào đọc từ FD 0 thì nó sẽ lấy dữ liệu từ file test2.txt
- Khi dùng < thì test2.txt -> FD 0 -> sort -> FD 1 -> Màn hình

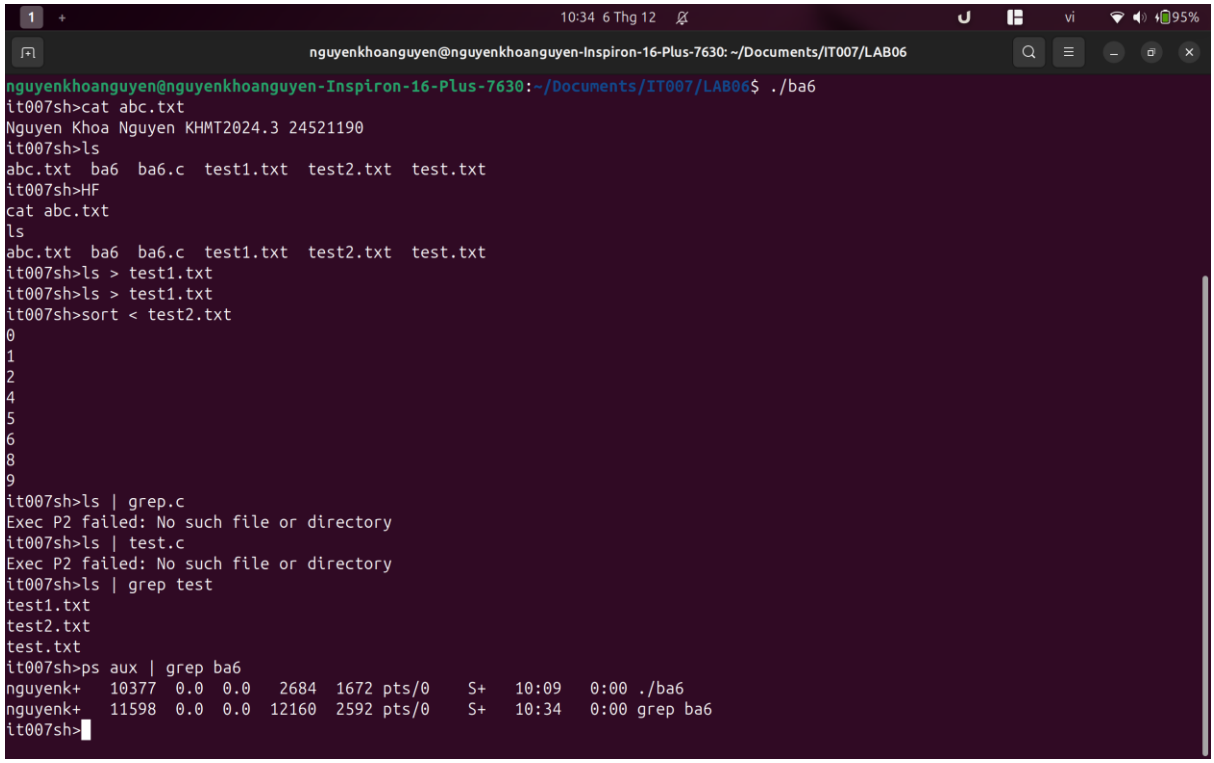
Câu 4: Giao tiếp sử dụng cơ chế đường ống Cho phép đầu ra của một lệnh đóng vai trò là đầu vào cho lệnh khác bằng cách sử dụng một đường ống. Ví dụ: it007sh>ps aux | grep firefox. Có đầu ra của lệnh ps aux đóng vai trò là đầu vào cho lệnh grep firefox . Gợi ý: sử dụng hàm pipe() và dup2()

- Testcase 1: Lọc tên file (ls | grep)



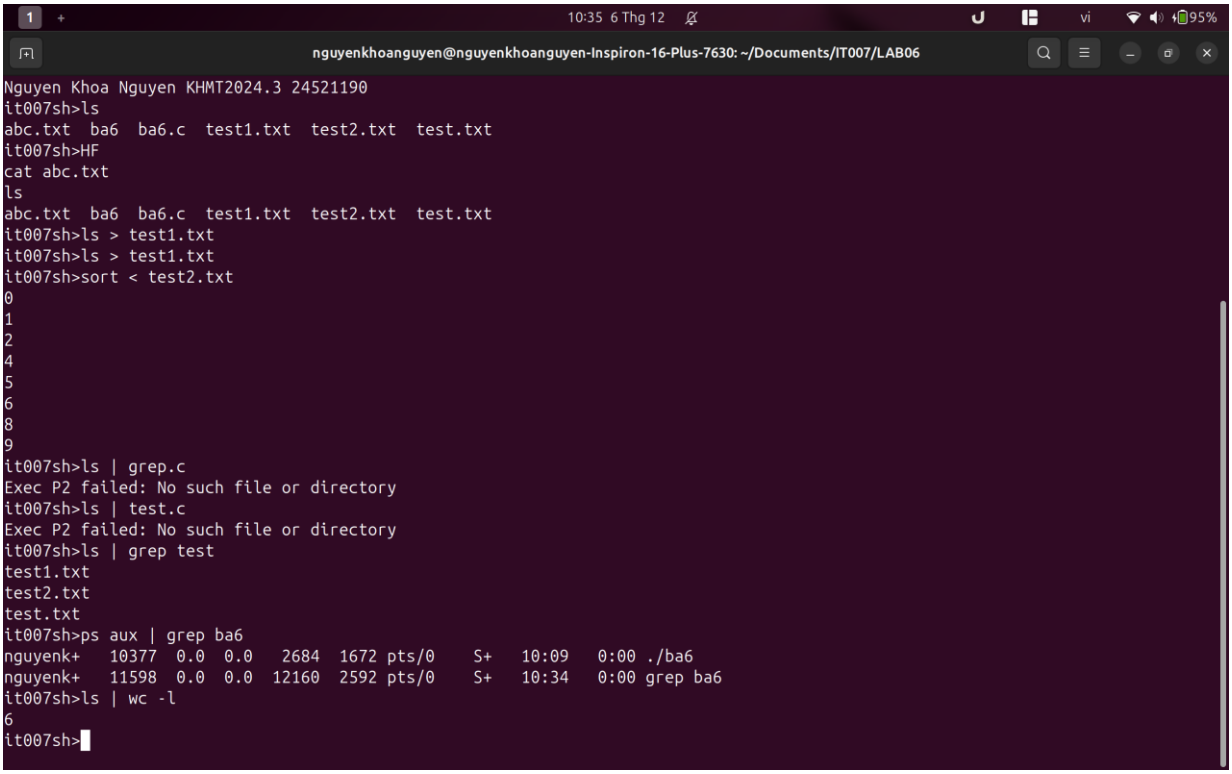
Hình 17 Thực thi lệnh ls | grep test

- Testcase 2: Kiểm tra tiến trình (ps | grep): Hiện ra thông tin về tiến trình (PID, User, Time....) đang chạy



Hình 18 Thực thi lệnh ps | grep

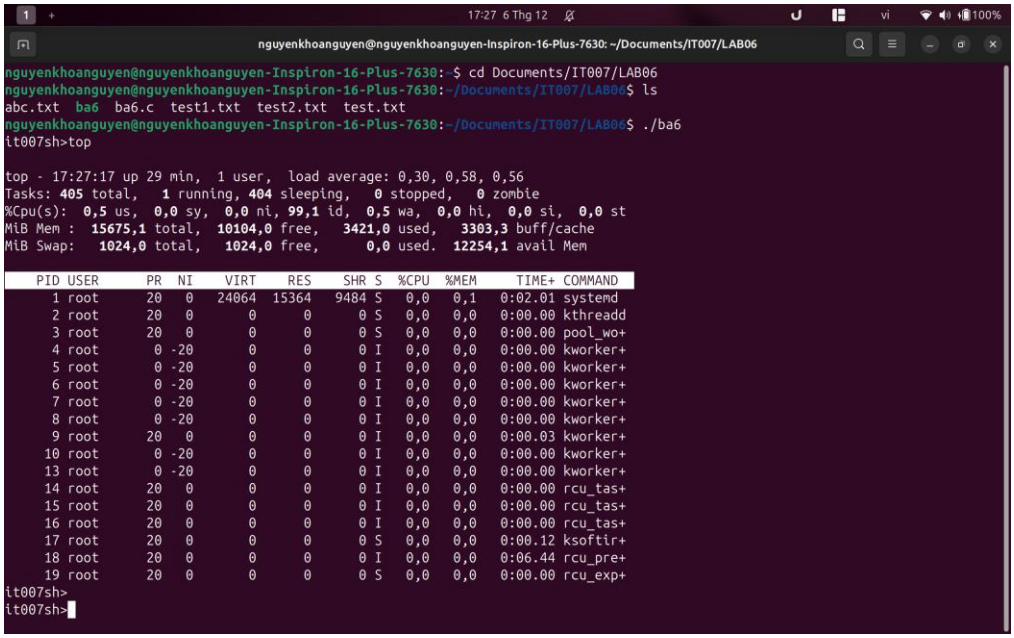
- Testcase : Đếm số lượng (ls | wc): Hiện ra tổng số file và thư mục đang có trong folder hiện tại



Hình 19 Thực thi lệnh ls | wc

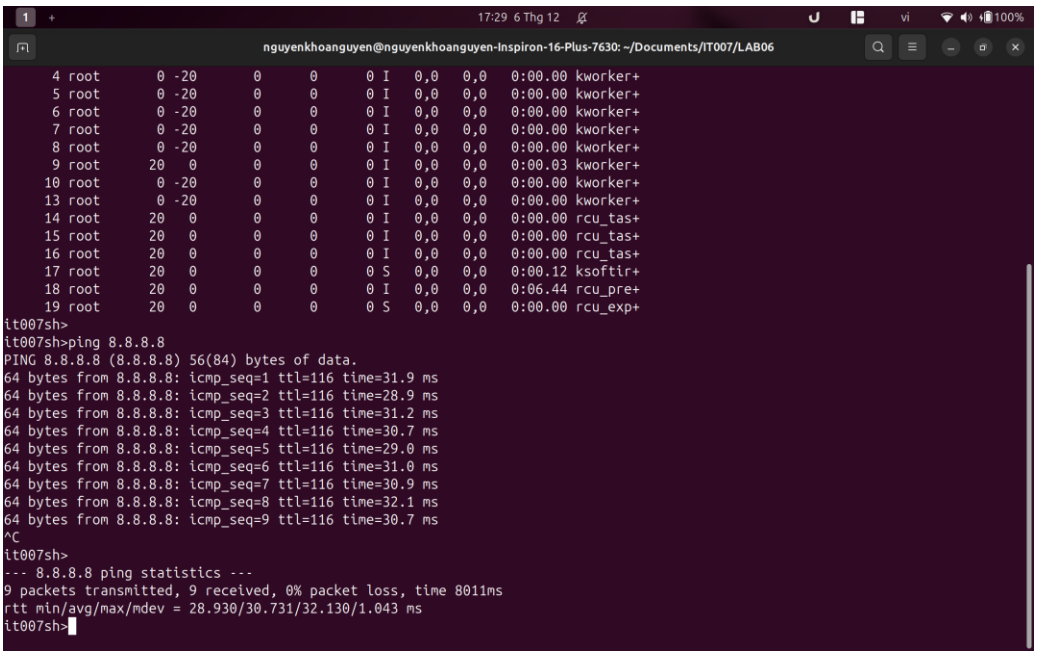
Câu 5: Kết thúc lệnh đang thực thi. Ví dụ: khi thực hiện lệnh it007sh>top kết quả sẽ liên tục hiển thị các process của hệ thống, khi sử dụng tổ hợp phím Ctrl + C, lệnh thực thi trên sẽ kết thúc và hiển thị dấu nhắc it007sh> mời người dùng nhập lệnh tiếp theo.

- Ngắt lệnh top: Khi ta nhấn Ctrl + C, lệnh top bị ngắt và xuất hiện dòng cho phép người dùng nhập lệnh mới



Hình 20 Kết quả khi ta ngắt lệnh top

- Ngắt lệnh ping: Lệnh ping trên Linux sẽ chạy mãi mãi cho đến khi bị ngắt, do đó ta kiểm tra như sau:



Hình 21 Kết quả khi ta ngắt lệnh ping 8.8.8.8