Báo cáo thực hành môn Hệ điều hành - Giảng viên: Trần Hoàng Lộc.

Họ và tên: Phạm Nguyên Anh, Phan Hoàng Mạnh Cường , Phạm Đăng Hoàng
Mã số sinh viên: 22520069, 22520179, 22520472
Lớp: IT007.O15.1

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	☒	☒	☒	☒	☒
Chụp hình minh chứng (chạy ít nhất 3 lệnh)	☒	☒	☒	☒	☒
Giải thích code, kết quả	☒	☒	☒	☒	☒

Tự chấm điểm: 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

1. Câu 1 và Câu 2 :

1. *Thực thi command trong tiến trình con.*
2. *Tạo tính năng sử dụng lại các câu lệnh trong quá khứ*

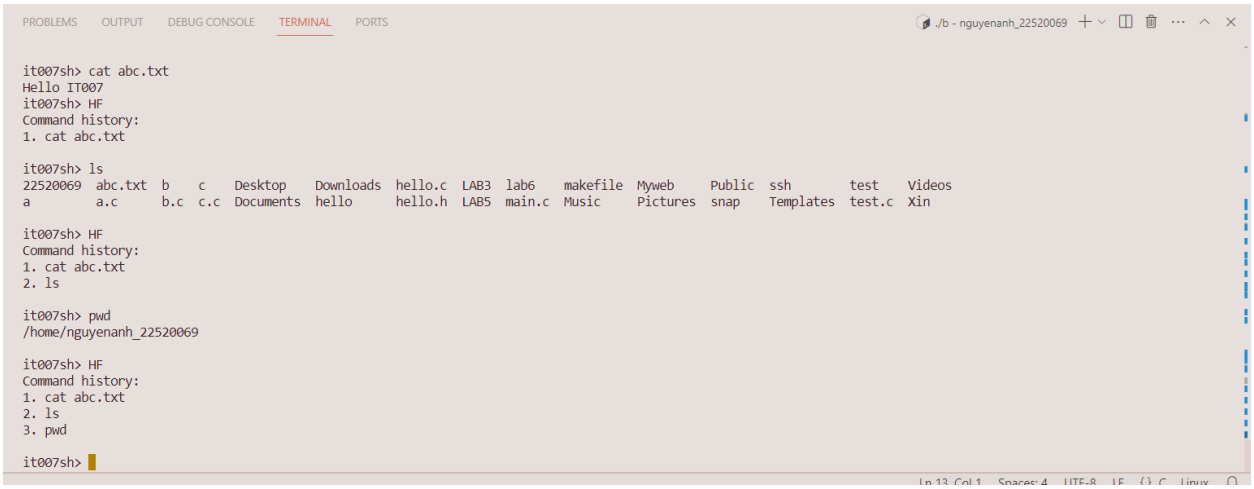
Code :

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

#define MAX_HISTORY_SIZE 10
#define MAX_INPUT_SIZE 100

char history[MAX_HISTORY_SIZE][MAX_INPUT_SIZE];
int historyIndex = 0;

void execute_command(char *command)
{
    strncpy(history[historyIndex % MAX_HISTORY_SIZE], command, MAX_INPUT_SIZE -
1);
    historyIndex++;
    pid_t pid = fork();

    if (pid < 0)
    {
        perror("Fork failed");
    }
    else if (pid == 0)
    {
        char *args[MAX_INPUT_SIZE];
        int i = 0;
        char *token = strtok(command, " ");
        while (token != NULL)
        {
            args[i++] = token;
            token = strtok(NULL, " ");
        }
        args[i] = NULL;
        execvp(args[0], args);
    }
```

```
    else
    {
        waitpid(pid, NULL, 0);
    }
}

void display_history ()
{
    printf("Command history:\n");
    int i;
    for (i = 0; i < MAX_HISTORY_SIZE; i++)
    {
        if (strlen(history[i]) > 0)
        {
            printf("%d. %s\n", i + 1, history[i]);
        }
    }
}
void print_prompt()
{
    printf("\nit007sh> ");
}

int main()
{
    char input[MAX_INPUT_SIZE];

    while (1)
    {
        print_prompt();
        fgets(input, MAX_INPUT_SIZE, stdin);
        input[strcspn(input, "\n")] = '\0';

        if (strcmp(input, "HF") == 0)
        {
            display_history();
        }
        else
        {
            execute_command(input);
        }
    }
    return 0;
}
```

Trình bày giải thuật :

- Tạo hàm execute_command để thực hiện lệnh được nhập.
- Tạo hàm display_history để hiển thị lịch sử các câu lệnh được nhập.
- Tạo hàm print_prompt để in ra dấu nhắc.
- Hàm main trong vòng lặp while(1) thực hiện các câu lệnh :
    - Nhập lệnh.
    - Nếu “HF” được nhập thì sẽ in lịch sử.
    - Nếu lệnh khác được nhập thì sẽ thực hiện lệnh đó.



Giải thích code, kết quả :

- *execute_command(char command):
    - Thực hiện lệnh nhập từ người dùng.
    - Lưu trữ lệnh vào mảng lịch sử.
    - Sử dụng fork để tạo một tiến trình con.
    - Trong tiến trình con, sử dụng execvp để thực hiện lệnh nhập từ người dùng.
    - Trong tiến trình cha, sử dụng waitpid để đợi tiến trình con kết thúc.
- display_history():
    - Hiển thị lịch sử các lệnh đã nhập.
- print_prompt():
    - In ra dấu nhắc để người dùng nhập lệnh.
- main():
    - Sử dụng vòng lặp vô hạn để liên tục đọc và thực thi lệnh từ người dùng.

- Nếu lệnh là "HF" (history), hiển thị lịch sử bằng cách gọi hàm displayHistory().
- Ngược lại, thực hiện lệnh bằng cách gọi hàm execute_command(input).

2. Câu 3 và Câu 4

Câu 3: Chuyển hướng vào ra

Code:

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

#define BUFFER_SIZE 1024

int main() {
    char cmd[BUFFER_SIZE];
    char *cmd_tokens[10];
    pid_t pid;

    while (1) {
        printf("it007sh>");
        fgets(cmd, BUFFER_SIZE, stdin);
        cmd[strlen(cmd) - 1] = '\0';

        // Check if the user wants to exit
        if (strcmp(cmd, "exit") == 0) {
            break;
        }

        // Tách lệnh thành các đối số
        int i = 0;
        char *token = strtok(cmd, " ");
        while (token != NULL) {
            cmd_tokens[i++] = token;
            token = strtok(NULL, " ");
        }
        cmd_tokens[i] = NULL;
```

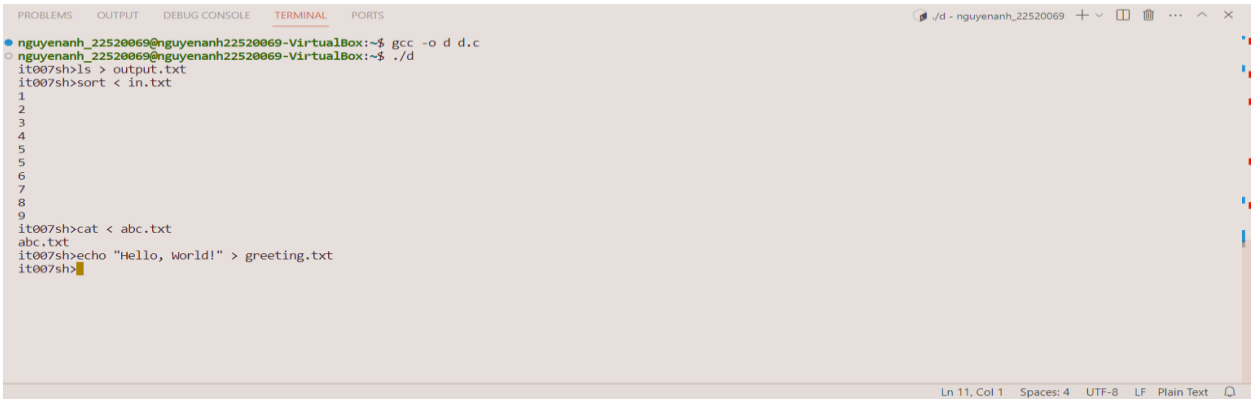
```
        // Tìm toán tử redirection
        int redirect_input = 0;
        int input_index = -1;
        int redirect_output = 0;
        int output_index = -1;

        for (int j = 0; j < i; ++j) {
            if (strcmp(cmd_tokens[j], "<") == 0) {
                redirect_input = 1;
                input_index = j;
            }
            if (strcmp(cmd_tokens[j], ">") == 0) {
                redirect_output = 1;
                output_index = j;
            }
        }

        // Fork child process
        pid = fork();

        if (pid == 0) {
            // Child process

            // Chuyển hướng đầu vào nếu có
            if (redirect_input) {
                int input_fd = open(cmd_tokens[input_index + 1], O_RDONLY);
                if (input_fd == -1) {
                    perror("open");
                    exit(1);
                }
                dup2(input_fd, STDIN_FILENO);
                close(input_fd);

                // Loại bỏ toán tử chuyển hướng và tên tệp từ danh sách đối số
                for (int k = input_index; k < i - 2; ++k) {
                    cmd_tokens[k] = cmd_tokens[k + 2];
                }
                cmd_tokens[i - 2] = NULL;
                cmd_tokens[i - 1] = NULL;
            }

            // Chuyển hướng đầu ra nếu có
            if (redirect_output) {
```

```
                int output_fd = open(cmd_tokens[output_index + 1], O_WRONLY |
O_CREAT | O_TRUNC, 0666);
                if (output_fd == -1) {
                    perror("open");
                    exit(1);
                }
                dup2(output_fd, STDOUT_FILENO);
                close(output_fd);

                // Loại bỏ toán tử chuyển hướng và tên tệp từ danh sách đối số
                for (int k = output_index; k < i - 2; ++k) {
                    cmd_tokens[k] = cmd_tokens[k + 2];
                }
                cmd_tokens[i - 2] = NULL;
                cmd_tokens[i - 1] = NULL;
            }

            // Thực hiện lệnh
            execvp(cmd_tokens[0], cmd_tokens);
            perror("execvp");
            exit(1);
        } else {
            // Parent process
            wait(NULL);
        }
    }

    return 0;
}
```

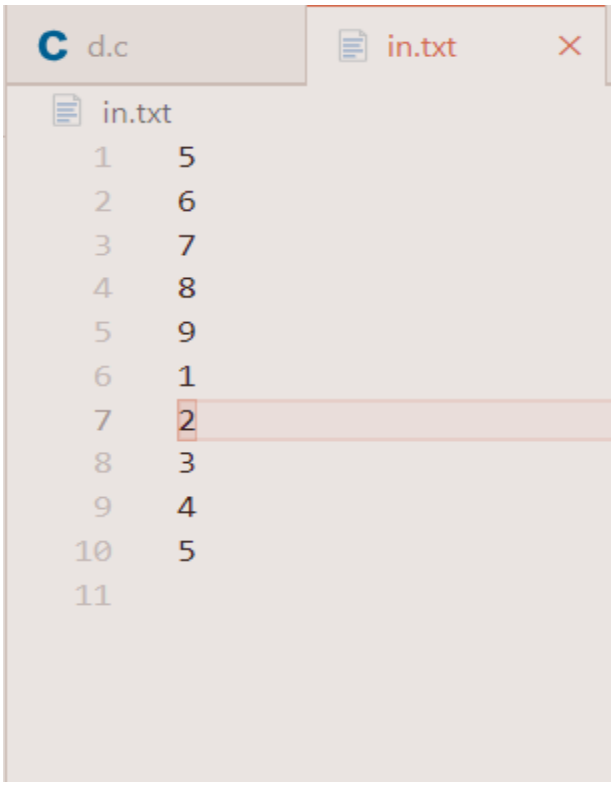
Trình bày giải thuật :

- Sử dụng hàm fork() để tạo một tiến trình con chạy lệnh
- Sử dụng hàm execvp() để thực thi lệnh trong tiến trình con.
- Hàm wait() được sử dụng để đảm bảo tiến trình cha không kết thúc trước tiến trình con.
- Sử dụng các hàm như open(), creat(), dup2(), và close() để thực hiện chuyển hướng đầu vào/đầu ra.

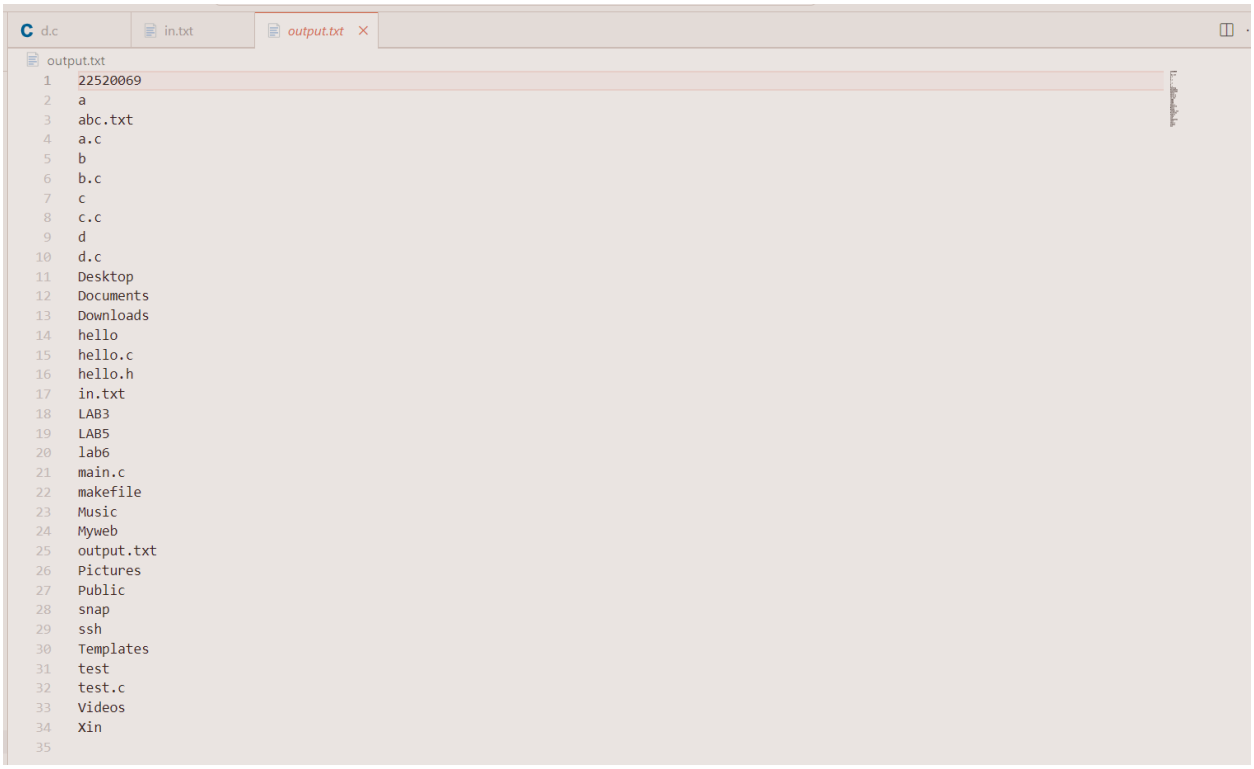
Kết quả chạy code :



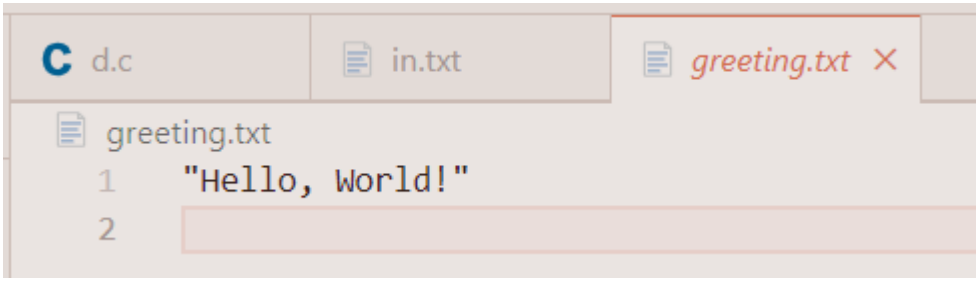
Thực hiện lệnh ls rồi gán vào file output.txt, sort file in.txt, gán Hello, world! vào greeting.txt



File in.txt chưa được sort.



File output.txt được tạo với nội dụng khi thực hiện của lệnh ls



File greeting.txt được tạo với nội dung "Hello, world!"

Giải thích code, kết quả :

1. Chương trình này sẽ hiển thị dấu nhắc it007sh> và chờ người dùng nhập vào một lệnh.
2. Nếu người dùng nhập exit, chương trình sẽ kết thúc.
3. Nếu không, chương trình sẽ tách lệnh thành các đối số bằng cách sử dụng khoảng trắng làm dấu phân cách.
4. Chương trình sau đó tìm kiếm các toán tử chuyển hướng (< và >). Nếu tìm thấy, nó sẽ ghi nhớ vị trí của chúng và tệp liên quan.
5. Chương trình sau đó tạo một tiến trình con bằng cách sử dụng fork().
6. Trong tiến trình con:

- Nếu có toán tử chuyển hướng đầu vào (<), nó mở tệp đầu vào, chuyển hướng STDIN đến tệp này, và loại bỏ toán tử chuyển hướng và tên tệp khỏi danh sách đối số.
- Nếu có toán tử chuyển hướng đầu ra (>), nó mở (hoặc tạo) tệp đầu ra, chuyển hướng STDOUT đến tệp này, và loại bỏ toán tử chuyển hướng và tên tệp khỏi danh sách đối số.
- Cuối cùng, nó thực hiện lệnh bằng cách sử dụng execvp().

7. Trong tiến trình cha, nó chờ đợi tiến trình con kết thúc bằng cách sử dụng wait().

Câu 4: Giao tiếp sử dụng cơ chế đường ống

Code:

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

#define BUFFER_SIZE 1024

int main() {
    char cmd[BUFFER_SIZE];
    char *cmd_tokens[10];
    pid_t pid;
    int pipe_fd[2];

    while (1) {
        printf("it007sh>");
        fgets(cmd, BUFFER_SIZE, stdin);
        cmd[strlen(cmd) - 1] = '\0';

        if (strcmp(cmd, "exit") == 0) {
            break;
        }

        int i = 0;
        char *token = strtok(cmd, " ");
        while (token != NULL) {
            cmd_tokens[i++] = token;
            token = strtok(NULL, " ");
        }
```

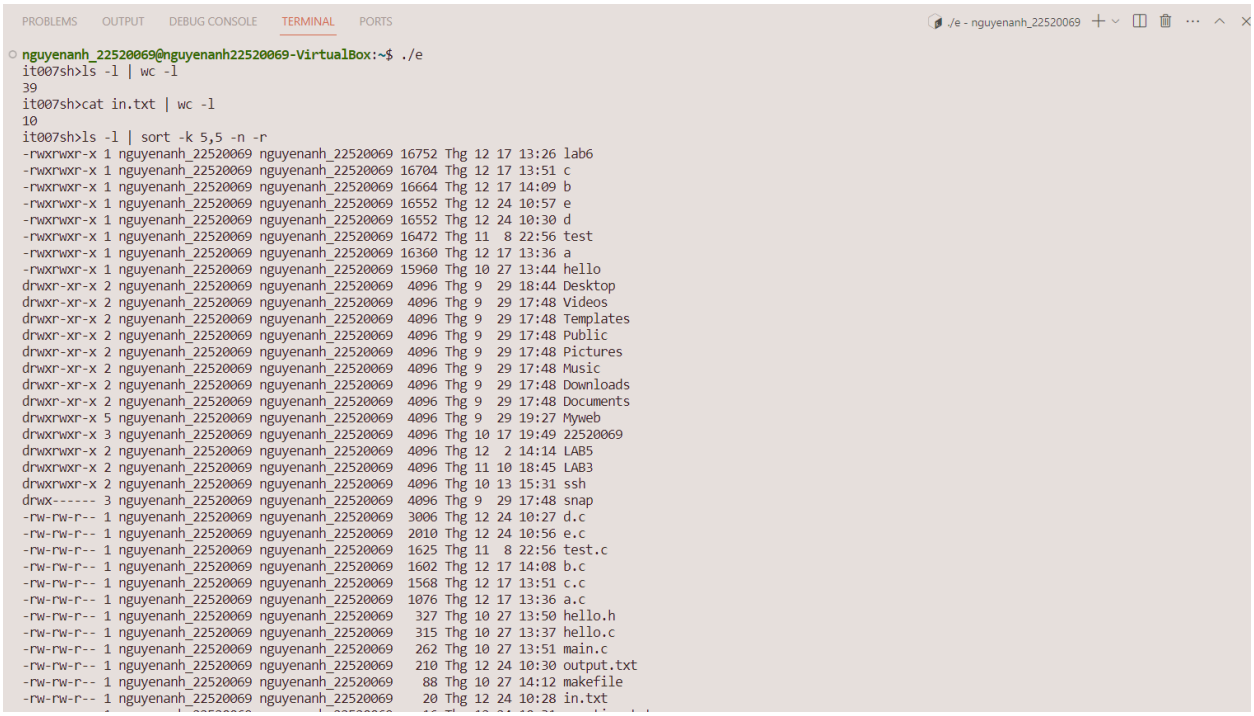
```
        cmd_tokens[i] = NULL;

        int pipe_index = -1;

        for (int j = 0; j < i; ++j) {
            if (strcmp(cmd_tokens[j], "|") == 0) {
                pipe_index = j;
                break;
            }
        }

        pid = fork();

        if (pid == 0) {
            if (pipe_index != -1) {
                if (pipe(pipe_fd) == -1) {
                    perror("pipe");
                    exit(1);
                }

                pid_t pid2 = fork();

                if (pid2 == 0) {
                    close(pipe_fd[0]);

                    dup2(pipe_fd[1], STDOUT_FILENO);

                    cmd_tokens[pipe_index] = NULL;
                    execvp(cmd_tokens[0], cmd_tokens);
                    perror("execvp");
                    exit(1);
                } else if (pid2 > 0) {
                    close(pipe_fd[1]);

                    dup2(pipe_fd[0], STDIN_FILENO);

                    execvp(cmd_tokens[pipe_index + 1], &cmd_tokens[pipe_index +
1]);
                    perror("execvp");
                    exit(1);
                } else {
                    perror("fork");
                    exit(1);
                }
```

```
            } else {
                execvp(cmd_tokens[0], cmd_tokens);
                perror("execvp");
                exit(1);
            }
        } else {
            wait(NULL);
        }
    }

    return 0;
}
```

Trình bày giải thuật:

- Sử dụng hàm fork() để tạo một tiến trình con chạy lệnh
- Hàm system() để thực thi lệnh trong tiến trình con
- Hàm wait() để đảm bảo tiến trình cha không kết thúc trước tiến trình con
- Sử dụng các hàm như pipe(), dup2() và close() để thực hiện chuyển hướng đầu vào/đầu ra thông qua đường ống.

Kết quả chạy code :



Thực hiện việc đếm số lượng tệp tin trong thư mục hiện tại, đếm số dòng trong một tệp tin văn bản và hiển thị danh sách các tệp tin và thư mục, sau đó sắp xếp theo kích thước giảm dần.

Giải thích code, kết quả :

1. Chương trình này sẽ hiển thị dấu nhắc it007sh> và chờ người dùng nhập vào một lệnh.
2. Nếu người dùng nhập exit, chương trình sẽ kết thúc.
3. Nếu không, chương trình sẽ tách lệnh thành các đối số bằng cách sử dụng khoảng trắng làm dấu phân cách.
4. Chương trình sau đó tìm kiếm toán tử ống (|). Nếu tìm thấy, nó sẽ ghi nhớ vị trí của nó.
5. Chương trình sau đó tạo một tiến trình con bằng cách sử dụng fork().
6. Trong tiến trình con:
    - Nếu có toán tử ống (|), nó tạo một ống bằng cách sử dụng pipe(), sau đó tạo một tiến trình con khác.
        - Trong tiến trình con thứ hai, nó đóng đầu đọc của ống, chuyển hướng STDOUT đến đầu ghi của ống, và thực hiện phần lệnh trước toán tử ống.
        - Trong tiến trình con đầu tiên, nó đóng đầu ghi của ống, chuyển hướng STDIN đến đầu đọc của ống, và thực hiện phần lệnh sau toán tử ống.
    - Nếu không có toán tử ống, nó chỉ đơn giản thực hiện lệnh bằng cách sử dụng execvp().
7. Trong tiến trình cha, nó chờ đợi tiến trình con kết thúc bằng cách sử dụng wait().

3. Câu 5

Code:

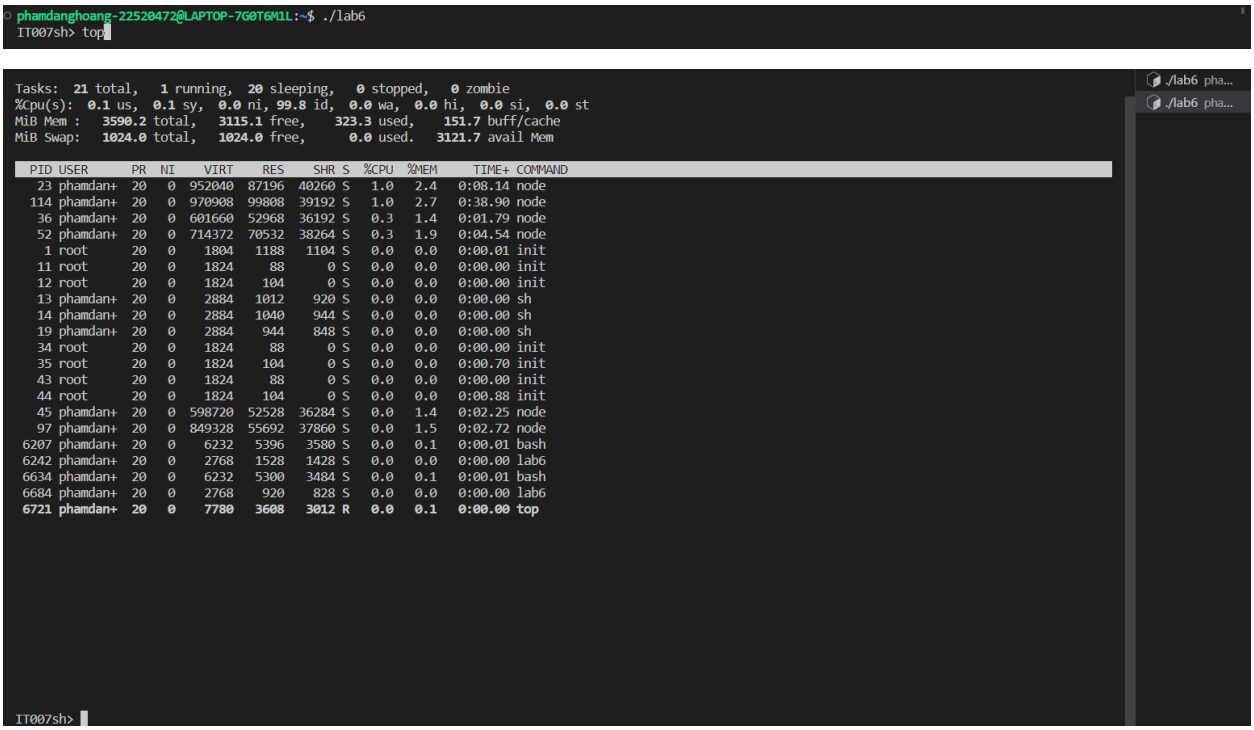
lab6.c

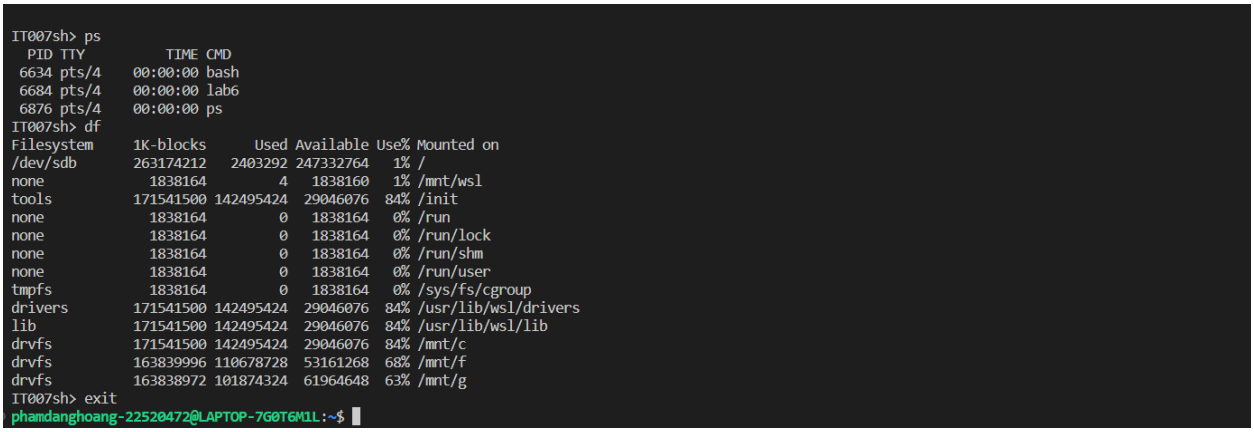
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>

#define MAX_LINE 80

volatile sig_atomic_t child_running = 0;

void handle_sigchld(int sig) {
    child_running = 0;
}

void handle_sigint(int sig) {

}

int main(void) {
    char input[MAX_LINE];

    signal(SIGINT, handle_sigint);

    while (1) {
        if (!child_running) {
            signal(SIGCHLD, handle_sigchld);
        }

        printf("IT007sh> ");
        fflush(stdout);

        if (fgets(input, sizeof(input), stdin) == NULL) {
            break;
        }


        input[strcspn(input, "\n")] = '\0';

        if (strcmp(input, "exit") == 0) {
            break;
        }

        pid_t pid = fork();

        if (pid < 0) {
            fprintf(stderr, "Fork failed\n");
            return 1;
        } else if (pid == 0) {

            execlp(input, input, (char *)NULL);
            perror("execlp");
            exit(EXIT_FAILURE);
        } else {

            child_running = 1;

            wait(NULL);
            signal(SIGCHLD, SIG_DFL);
        }
    }

    return 0;
}
```

Trình bày giải thuật :

- Tạo hàm handle_sigint(int sig): Xử lý tín hiệu SIGINT (Ctrl+C). Có thể mở rộng để xử lý các tác vụ khi tín hiệu này được nhận.
- Tạo hàm handle_sigchld(int sig) và hàm sẽ được gọi khi một quy trình con kết thúc. Đặt child_running về 0 để thông báo rằng không có quy trình con đang chạy.
- Hàm main():
    - Tạo hàm với vòng lặp vô hạn chờ lệnh
    - Hiển thị dấu nhắc "IT007sh> ".
      Nếu lệnh là "exit", thoát khỏi vòng lặp và kết thúc chương trình.
    - Trong tiến trình con : Sử dụng execlp() để thực hiện lệnh đã nhập từ người dùng.
      Trong tiến trình cha : Chờ tiến trình con và đặt lại xử lý tín hiệu SIGCHLD về giá trị mặc định

Kết quả chạy code :





Giải thích kết quả:

- Chạy chương trình và sử dụng lệnh top hiển thị ra bảng thông kê các thông tin từ hệ thống
- Sử dụng lệnh ‘Ctrl+C’ lệnh thực thi sẽ kết thúc và mời người dung
- Tương tự sử dụng lệnh ps và df hiện thị thông tin về quy trình đang chạy và không gian đĩa sử dụng/còn trống trên hệ thống.
- Dùng lệnh exit để kết thúc chương trình.
- Tiến trình con (fork): Tiến trình con sẽ thực thi lệnh được nhập từ người dùng.
- Tiến trình cha đợi tiến trình con kết thúc (wait)
- Sau khi tiến trình con kết thúc, đặt child_running về 0 để báo hiệu rằng không có tiến trình con nào đang chạy.
- Sử dụng signal để đặt xử lý tín hiệu SIGCHLD về mặc định.
- Lặp lại quá trình nhập lệnh.
- Quay lại vòng lặp để chờ người dùng nhập lệnh mới.