|  |
| --- |
| Họ và tên: Dương Thuận Trí  Mã số sinh viên: 22521517  Lớp: 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**

# Câu 1

Trả lời:

* **Giải thuật:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#define MAX\_LINE 80

*int* main(*void*) {

*char* \*args[MAX\_LINE / 2 + 1];

*int* shouldrun = 1;

    while (shouldrun) {

        printf("it007sh>");

        fflush(stdout);

*char* command[MAX\_LINE]; // mang dung` de chua lenh

        fgets(command, MAX\_LINE, stdin);

*char* \*arg = strtok(command, " \n");

*int* i = 0;

        while (arg != NULL) {

            args[i++] = arg;

            arg = strtok(NULL, " \n");

        }

        args[i] = NULL;

*pid\_t* pid = fork();

        if (pid < 0) {

            fprintf(stderr, "Fork that bai!\n");

            return 1;

        } else if (pid == 0) {

            execvp(args[0], args);

            fprintf(stderr, "Khong tim thay lenh: %s\n", args[0]);

            exit(EXIT\_FAILURE);

        } else {

*int* status;

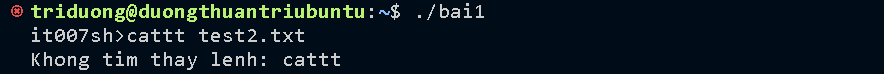
            waitpid(pid, &status, 0);

        }

    }

    return 0;

}

* **A black background with white text

  Description automatically generatedOutput:**
* **Giải thích:**
* Mảng args[MAX\_LINE / 2 + 1] để lưu trữ các đối số của lệnh. Số lượng đối số tối đa được giới hạn là (MAX\_LINE / 2) + 1.
* Mảng command[MAX\_LINE] để lưu trữ lệnh được nhập từ người dùng. Độ dài tối đa của lệnh được giới hạn bởi MAX\_LINE.
* Dùng hàm fgets(command, MAX\_LINE, stdin) để đọc lệnh từ người dùng và lưu vào mảng command
* Dùng strtok(command, " \n") để tách lệnh thành các đối số và lưu tạm vào chuỗi arg (strtok đầu tiên dùng để tách đối số đầu tiên- ví dụ như “ls”, “cat”,…), ký tự " " và "\n" là những ký tự phân tách (phân tách khi gặp khoảng trắng/ xuống dòng). Ta lưu từng đối số vào mảng args (args[0] chính là phần tử chứa đối số lệnh)
* “pid\_t pid = fork()”: Tạo một tiến trình con.
* pid<0: fork thất bại, in thông báo lỗi và thoát.
* pid==0: đây là tiến trình con, sử dụng execvp(args[0], args) để thực thi lệnh. Nếu execvp thất bại, in thông báo lỗi và thoát với mã lỗi.
* pid>0: đây là tiến trình cha, đợi tiến trình con kết thúc và lưu trữ trạng thái.
* Giải thích kết quả:
* “ls”: hiển thị danh sách các file/thư mục ở thư mục hiện tại
* “cat test2.txt”: hiển thị nội dung file test2.txt
* “cp test2.txt test3.txt”: sao chép test2.txt thành một file mới có tên test3.txt ngay tại thư mục hiện tại.
* “cat test3.txt”: hiển thị nội dung file test3.txt ( có cùng nội dung với file test2.txt vì đây là một file sao chép từ test2.txt)
* “cattt test2.txt”: không tồn tại lệnh này nên in thông báo lỗi

# Câu 2

Trả lời:

* **Giải thuật:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#define MAX\_LINE 80

#define HF\_size 20 // so cau lenh toi da duoc luu trong lich su

*char* history[HF\_size][MAX\_LINE];

// dung mang de luu lich su nhung cau lenh da nhap

*int* his\_index = 0;

*void* add\_to\_history(const *char* \**command*) {

    strcpy(history[his\_index], command);

    his\_index = (his\_index + 1) % HF\_size;

    // thay cau lenh cu~ nhat bang cau lenh moi nhat neu da dat so cau lenh toi //da duoc luu

}

*void* print\_history() {

*int* i = 0;

*int* print\_index = his\_index;

    do {

        if (strlen(history[print\_index]) > 0) {

            printf("%s\n", history[print\_index]);

        }

        print\_index = (print\_index + 1) % HF\_size;

        i++;

    } while (i < HF\_size);

}

*int* main(*void*) {

*char* \*args[MAX\_LINE / 2 + 1];

*int* shouldrun = 1;

    while (shouldrun) {

        printf("it007sh>");

        fflush(stdout);

*char* command[MAX\_LINE]; // mang dung` de chua lenh

        fgets(command, MAX\_LINE, stdin);

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

        if (strcmp(command, "HF") == 0)

        {

            print\_history(); // in ra truoc khi luu "HF" vao lich su

            add\_to\_history(command);

            continue;  //sang vong lap tiep theo

        }

        add\_to\_history(command);

*char* \*arg = strtok(command, " \n");

*int* i = 0;

        while (arg != NULL) {

            args[i++] = arg;

            arg = strtok(NULL, " \n");

        }

        args[i] = NULL;

*pid\_t* pid = fork();

        if (pid < 0) {

            fprintf(stderr, "Fork that bai!\n");

            return 1;

        } else if (pid == 0) {

            execvp(args[0], args);

            fprintf(stderr, "Khong tim thay lenh: %s\n", args[0]);

            exit(EXIT\_FAILURE);

        } else {

*int* status;

            waitpid(pid, &status, 0);

        }

    }

    return 0;

}

* A screen shot of a computer

  Description automatically generatedA screen shot of a computer

  Description automatically generated**Output:**
* **Giải thích:**
* Dùng mảng history[HF\_size][MAX\_LINE] để lưu những câu lệnh đã nhập (HF\_size chính là số lượng tối đa câu lệnh được lưu, nếu bị quá số lượng tối đa, câu lệnh mới nhất sẽ thay thế câu lệnh cũ nhất trong hàm add\_to\_history)
* Hàm add\_to\_history(const char\* command) để thêm lệnh mới vào lịch sử, ghi đè lệnh cũ nhất nếu đã đạt số lệnh tối đa được lưu trong lịch sử.
* Hàm print\_history() để in ra lịch sử câu lệnh (HF\_size hoặc nhỏ hơn HF\_size câu lệnh gần nhất)
* Điều kiện if (strcmp(command, "HF") == 0): xét nếu người dùng nhập "HF", chương trình sẽ gọi print\_history() trước khi gọi add\_to\_history(command) để in ra lịch sử trước khi lưu "HF" vào lịch sử. “continue” sang vòng lập kế tiếp để người dùng nhập câu lệnh khác.
* Những giải thuật khác để thực thi lệnh khác lệnh “HF” (tương tự yêu cầu 1)
* Giải thích kết quả:
* HF đầu tiên: in ra “ls” và “cat test2.txt” vì vừa nhập 2 câu lệnh đó.
* Những HF sau tương tự: in ra những câu lệnh đã nhập vào trước đó kể cả lệnh nhập sai và lệnh HF.

1. **Câu 3**

Trả lời:

* **Giải thuật:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <signal.h>

#define MAX\_LINE 80

#define HF\_size 20 // so cau lenh toi da duoc luu trong lich su

*char* history[HF\_size][MAX\_LINE];

*int* his\_index = 0;

*pid\_t* running\_pid;

*void* add\_to\_history(const *char* \**command*) {

    strcpy(history[his\_index], command);

    his\_index = (his\_index + 1) % HF\_size;

    // thay cau lenh cu~ nhat bang cau lenh moi nhat neu da dat so cau lenh toi da duoc luu

}

*void* print\_history() {

*int* i = 0;

*int* print\_index = his\_index;

    do {

        if (strlen(history[print\_index]) > 0) {

            printf("%s\n", history[print\_index]);

        }

        print\_index = (print\_index + 1) % HF\_size;

        i++;

    } while (i < HF\_size);

}

// ham de xu li command binh thuong

*void* execute\_command(*char* \**args*[]) {

*pid\_t* pid = fork();

    if (pid < 0) {

        fprintf(stderr, "Fork that bai!\n");

        exit(EXIT\_FAILURE);

    } else if (pid == 0) {

        execvp(args[0], args);

        fprintf(stderr, "Khong tim thay lenh: %s\n", args[0]);

        exit(EXIT\_FAILURE);

    } else {

        running\_pid = pid;

*int* status;

        waitpid(pid, &status, 0);

        running\_pid = 0;

    }

}

// ham de xu li dau` ra/vao

*void* execute\_redirection(*char* \**args*[], *int* *input\_fd*, *int* *output\_fd*) {

*pid\_t* pid = fork();

    if (pid < 0) {

        fprintf(stderr, "Fork that bai!\n");

        exit(EXIT\_FAILURE);

    } else if (pid == 0) {

        if (input\_fd != STDIN\_FILENO) { //dau vao`

            dup2(input\_fd, STDIN\_FILENO);

            close(input\_fd);

        }

        if (output\_fd != STDOUT\_FILENO) { //dau ra

            dup2(output\_fd, STDOUT\_FILENO);

            close(output\_fd);

        }

        execvp(args[0], args);

        perror("execvp");

        exit(EXIT\_FAILURE);

    } else {

*int* status;

        waitpid(pid, &status, 0);

    }

}

*int* main(*void*) {

*char* \*args[MAX\_LINE / 2 + 1];

*int* shouldrun = 1;

    while (shouldrun) {

        printf("it007sh>");

        fflush(stdout);

*char* command[MAX\_LINE]; // mang dung` de chua lenh

        fgets(command, MAX\_LINE, stdin);

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

        if (strcmp(command, "HF") == 0)

        {

            print\_history(); // in ra truoc khi luu "HF" vao lich su

            add\_to\_history(command);

            continue;  //sang vong lap tiep theo

        }

        add\_to\_history(command);

*char* \*arg = strtok(command, " \n");

*int* i = 0;

        while (arg != NULL) {

            args[i++] = arg;

            arg = strtok(NULL, " \n");

        }

        args[i] = NULL;

        // 2 bien output va input de dat co` khi co chuyen dau ra/vao ('>' '<')

*int* output = 0;

*int* input = 0;

        for (*int* j = 0; j < i; j++) {

            if (strcmp(args[j], ">") == 0) {

                output = 1;

                args[j] = NULL;

*int* fd = open(args[j + 1], O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

                if (fd == -1) {

                    perror("open");

                    exit(EXIT\_FAILURE);

                }

                execute\_redirection(args, STDIN\_FILENO, fd); // tach thanh` ham` //rieng de gon gang hon

                break;

            } else if (strcmp(args[j], "<") == 0) {

                input = 1;

                args[j] = NULL;

*int* fd = open(args[j + 1], O\_RDONLY);

                if (fd == -1) {

                    perror("open");

                    exit(EXIT\_FAILURE);

                }

                execute\_redirection(args, fd, STDOUT\_FILENO);

                break;

            }

        }

        // khi khong co chuyen huong dau ra/vao thi xu li binh thuong`

        if (!output && !input) {

            execute\_command(args); // tach thanh` ham` rieng de gon gang hon

        }

    }

    return 0;

}

* A screen shot of a computer

  Description automatically generated**Output:**

A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

* **Giải thích:**
* Vòng lặp for (int j = 0; j < i; j++) { ... } để quét qua mảng các đối số của lệnh tìm xem có kí tự ‘>’ hay ‘<’ không.
* Chuyển hướng đầu ra (>):
* Nếu tìm thấy đối số là >, đặt cờ output thành 1, loại bỏ dấu > khỏi danh sách đối số (args[j] = NULL).
* Mở một file descriptor (fd) cho đầu ra (open(args[j + 1], O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH)).
* Gọi hàm execute\_redirection để thực hiện lệnh với chuyển hướng đầu ra.
* Chuyển hướng đầu vào (<) tương tự:
* Nếu tìm thấy đối số là <, đặt cờ input thành 1, loại bỏ dấu < khỏi danh sách đối số.
* Mở một file descriptor (fd) cho đầu vào (open(args[j + 1], O\_RDONLY);).
* Gọi hàm execute\_redirection để thực hiện lệnh với chuyển hướng đầu vào.
* Hàm execute\_redirection: Đây là hàm thực hiện lệnh với chuyển hướng đầu vào và đầu ra. Nó nhận vào một mảng các đối số (args), một file descriptor cho đầu vào (input), và một file descriptor cho đầu ra (output). ( tùy vào tham số truyền vào để quyết định xử lí đầu vào/ra).
* Dùng fork() để tạo một tiến trình con để thực hiện lệnh.
* Tiến trình con (pid == 0):
* Điều kiện if (input\_fd != STDIN\_FILENO): xét nếu có chuyển hướng đầu vào, sử dụng dup2(input\_fd, STDIN\_FILENO) để sao chép file descriptor của đầu vào vào STDIN\_FILENO.
* Gọi close(input) để đóng file descriptor của đầu vào, vì nó đã được sao chép vào STDIN\_FILENO.
* Điều kiện if (output\_fd != STDOUT\_FILENO): xét nếu có chuyển hướng đầu ra, sử dụng dup2(output\_fd, STDOUT\_FILENO) để sao chép file descriptor của đầu ra vào STDOUT\_FILENO.
* Gọi close(output) đóng file descriptor của đầu ra, vì nó đã được sao chép vào STDOUT\_FILENO.
* execvp(args[0], args): Thực hiện lệnh với các đối số đã được truyền vào. Nếu lệnh thất bại, in ra thông báo lỗi và thoát với mã lỗi.
* Tiến trình cha: Đợi tiến trình con kết thúc. Trong khi đợi, chương trình cha bị đứng lại và không chạy lệnh tiếp theo cho đến khi tiến trình con hoàn thành.
* Giải thích kết quả:
  + “ls > test4.txt”: lưu output của lệnh ls vào file test4.txt vừa mới tạo ( chuyển hướng đầu ra)
  + “sort < test5.txt”: dùng nội dung của file test5.txt làm input của lệnh sort ( chuyển hướng đầu vào) -> in ra nội dung test5.txt đã được sắp xếp.
  + “cat test5.txt > test6.txt”: lưu nội dung của test5.txt vào test6.txt ( chuyển hướng đầu ra).

1. **Câu 4**

Trả lời:

* **Giải thuật:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <signal.h>

#define MAX\_LINE 80

#define HF\_size 20 // so cau lenh toi da duoc luu trong lich su

*char* history[HF\_size][MAX\_LINE];

*int* his\_index = 0;

*pid\_t* running\_pid;

*void* add\_to\_history(const *char* \**command*) {

    strcpy(history[his\_index], command);

    his\_index = (his\_index + 1) % HF\_size;

    // thay cau lenh cu~ nhat bang cau lenh moi nhat neu da dat so cau lenh toi da duoc luu

}

*void* print\_history() {

*int* i = 0;

*int* print\_index = his\_index;

    do {

        if (strlen(history[print\_index]) > 0) {

            printf("%s\n", history[print\_index]);

        }

        print\_index = (print\_index + 1) % HF\_size;

        i++;

    } while (i < HF\_size);

}

// ham de xu li command binh thuong

*void* execute\_command(*char* \**args*[]) {

*pid\_t* pid = fork();

    if (pid < 0) {

        fprintf(stderr, "Fork that bai!\n");

        exit(EXIT\_FAILURE);

    } else if (pid == 0) {

        execvp(args[0], args);

        fprintf(stderr, "Khong tim thay lenh: %s\n", args[0]);

        exit(EXIT\_FAILURE);

    } else {

        running\_pid = pid;

*int* status;

        waitpid(pid, &status, 0);

        running\_pid = 0;

    }

}

// ham de xu li dau` ra/vao

*void* execute\_redirection(*char* \**args*[], *int* *input*, *int* *output*) {

*pid\_t* pid = fork();

    if (pid < 0) {

        fprintf(stderr, "Fork that bai!\n");

        exit(EXIT\_FAILURE);

    } else if (pid == 0) {

        if (input != STDIN\_FILENO) { //dau vao`

            dup2(input, STDIN\_FILENO);

            close(input);

        }

        if (output != STDOUT\_FILENO) { //dau ra

            dup2(output, STDOUT\_FILENO);

            close(output);

        }

        execvp(args[0], args);

        perror("execvp");

        exit(EXIT\_FAILURE);

    } else {

*int* status;

        waitpid(pid, &status, 0);

    }

}

*void* execute\_pipeline(*char* \**args1*[], *char* \**args2*[]) {

*int* pipe\_fd[2];

    if (pipe(pipe\_fd) == -1) {

        perror("pipe");

        exit(EXIT\_FAILURE);

    }

*pid\_t* pid1 = fork();

    if (pid1 == 0) {

        close(pipe\_fd[0]);

        dup2(pipe\_fd[1], STDOUT\_FILENO);

        close(pipe\_fd[1]);

        execvp(args1[0], args1);

        perror("execvp");

        exit(EXIT\_FAILURE);

    } else if (pid1 > 0) {

        waitpid(pid1, NULL, 0);

*pid\_t* pid2 = fork();

        if (pid2 == 0) {

            close(pipe\_fd[1]);

            dup2(pipe\_fd[0], STDIN\_FILENO);

            close(pipe\_fd[0]);

            execvp(args2[0], args2);

            perror("execvp");

            exit(EXIT\_FAILURE);

        } else if (pid2 > 0) {

            close(pipe\_fd[0]);

            close(pipe\_fd[1]);

            waitpid(pid2, NULL, 0);

        } else {

            perror("fork");

            exit(EXIT\_FAILURE);

        }

    } else {

        perror("fork");

        exit(EXIT\_FAILURE);

    }

}

*int* main(*void*) {

*char* \*args[MAX\_LINE / 2 + 1];

*int* shouldrun = 1;

    while (shouldrun) {

        printf("it007sh>");

        fflush(stdout);

*char* command[MAX\_LINE]; // mang dung` de chua lenh

        fgets(command, MAX\_LINE, stdin);

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

        if (strcmp(command, "HF") == 0)

        {

            print\_history(); // in ra truoc khi luu "HF" vao lich su

            add\_to\_history(command);

            continue;  //sang vong lap tiep theo

        }

        add\_to\_history(command);

*char* \*arg = strtok(command, " \n");

*int* i = 0;

        while (arg != NULL) {

            args[i++] = arg;

            arg = strtok(NULL, " \n");

        }

        args[i] = NULL;

        // 2 bien output va input de dat co` khi co chuyen dau ra/vao ('>' '<')

*int* output = 0;

*int* input = 0;

*int* pipeline = 0;

        for (*int* j = 0; j < i; j++) {

            if (strcmp(args[j], ">") == 0) {

                output = 1;

                args[j] = NULL;

*int* fd = open(args[j + 1], O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

                if (fd == -1) {

                    perror("open");

                    exit(EXIT\_FAILURE);

                }

                execute\_redirection(args, STDIN\_FILENO, fd); // tach thanh` ham` rieng de gon gang hon

                break;

            } else if (strcmp(args[j], "<") == 0) {

                input = 1;

                args[j] = NULL;

*int* fd = open(args[j + 1], O\_RDONLY);

                if (fd == -1) {

                    perror("open");

                    exit(EXIT\_FAILURE);

                }

                execute\_redirection(args, fd, STDOUT\_FILENO);

                break;

            }

            else if (strcmp(args[j], "|") == 0) { // xet pipeline

                pipeline = 1;

                args[j] = NULL;

*char* \*args2[MAX\_LINE / 2 + 1];

*int* k = 0;

                for (*int* l = j + 1; l < i; l++) {

                    args2[k++] = args[l];

                }

                args2[k] = NULL;

                execute\_pipeline(args, args2);

                break;

            }

        }

        // xu li binh thuong khi khong co redirect, pipeline

        if (!output && !input && !pipeline) {

            execute\_command(args);

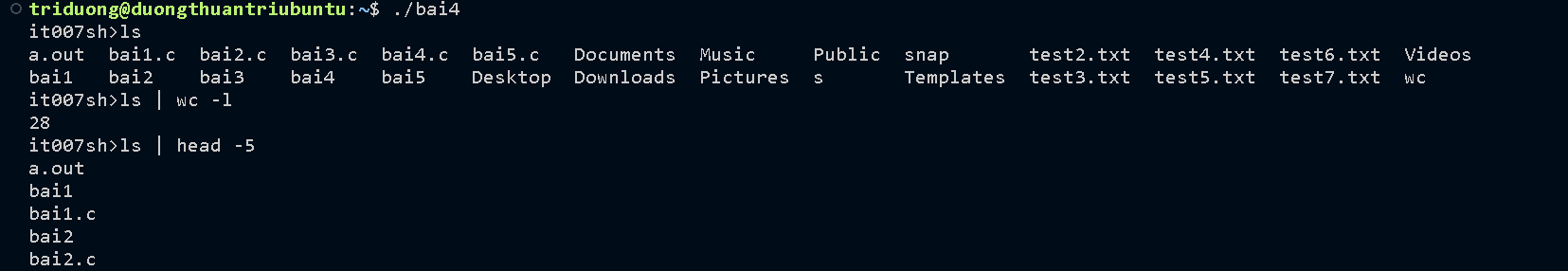
        }

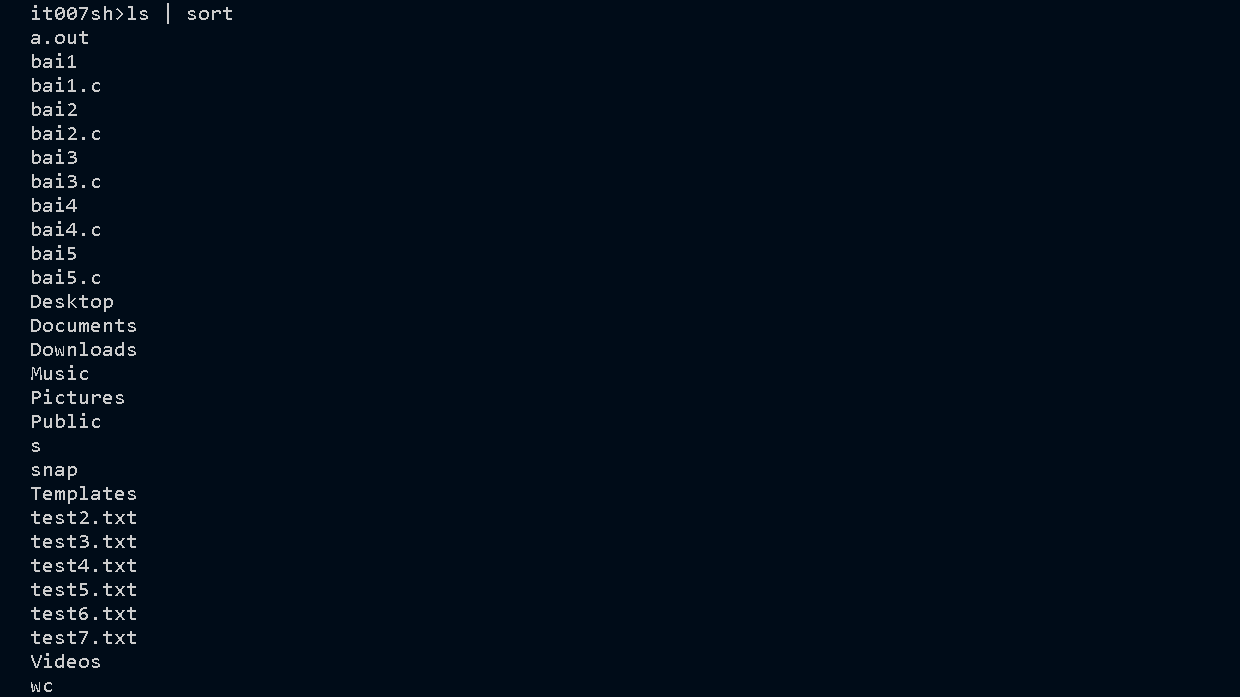
    }

    return 0;

}

* **Output:**





* **Giải thích:**
* Vòng lặp for (int j = 0; j < i; j++) { ... } để quét qua mảng các đối số của lệnh tìm xem có kí tự ‘|’ (pipeline) hay không. (‘<’ và ‘>’ xử lí tương tự yêu cầu 4)
* Nếu tìm thấy đối số là ‘|’, đặt cờ pipeline thành 1, loại bỏ dấu ‘|’ khỏi danh sách đối số (args[j]=NULL), sau đó tách lệnh thành hai phần, tạo một mảng mới để lưu lệnh thứ hai (args2[MAX\_LINE / 2 +1]) và gọi hàm execute\_pipeline(args, args2) để xử lý 2 phần lệnh.
* Hàm execute\_pipeline:
* Tạo ống (pipe\_fd): Sử dụng pipe để tạo một ống giữa hai tiến trình con.
* Tạo tiến trình con đầu tiên (pid1): Đóng đầu đọc của ống, thiết lập đầu ra của tiến trình con thành đầu vào của ống, và thực hiện lệnh đầu tiên.
* Chờ tiến trình con đầu tiên kết thúc: Tiến trình cha đợi cho tiến trình con đầu tiên kết thúc.
* Tạo tiến trình con thứ hai (pid2): Đóng đầu ghi của ống và thiết lập đầu vào của tiến trình con thành đầu ra của ống, sau đó thực hiện lệnh thứ hai.
* Chờ tiến trình con thứ hai kết thúc: Tiến trình cha đợi cho tiến trình con thứ hai kết thúc.
* Giải thích kết quả:
* “ls | wc -l”: liệt kê file, thư mục ở thư mục hiện tại và chuyển output đến lệnh wc -l để đếm số dòng.
* “ls | head -5”: liệt kê file, thư mục của thư mục hiện tại và chuyển output đến lệnh head -5 để chỉ hiển thị 5 dòng đầu tiên.
* “ls | sort”: tương tự và xuất ra output đã sắp xếp theo thứ tự từ điển.

1. **Câu 5**

Trả lời:

* **Giải thuật:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <signal.h>

#define MAX\_LINE 80

#define HF\_size 20 // so cau lenh toi da duoc luu trong lich su

*char* history[HF\_size][MAX\_LINE];

*int* his\_index = 0;

*pid\_t* running\_pid;

*void* add\_to\_history(const *char* \**command*) {

    strcpy(history[his\_index], command);

    his\_index = (his\_index + 1) % HF\_size;

    // thay cau lenh cu~ nhat bang cau lenh moi nhat neu da dat so cau lenh toi da duoc luu

}

*void* print\_history() {

*int* i = 0;

*int* print\_index = his\_index;

    do {

        if (strlen(history[print\_index]) > 0) {

            printf("%s\n", history[print\_index]);

        }

        print\_index = (print\_index + 1) % HF\_size;

        i++;

    } while (i < HF\_size);

}

// ham de xu li command binh thuong

*void* execute\_command(*char* \**args*[]) {

*pid\_t* pid = fork();

    if (pid < 0) {

        fprintf(stderr, "Fork that bai!\n");

        exit(EXIT\_FAILURE);

    } else if (pid == 0) {

        execvp(args[0], args);

        fprintf(stderr, "Khong tim thay lenh: %s\n", args[0]);

        exit(EXIT\_FAILURE);

    } else {

        running\_pid = pid;

*int* status;

        waitpid(pid, &status, 0);

        running\_pid = 0;

    }

}

// ham de xu li dau` ra/vao

*void* execute\_redirection(*char* \**args*[], *int* *input*, *int* *output*) {

*pid\_t* pid = fork();

    if (pid < 0) {

        fprintf(stderr, "Fork that bai!\n");

        exit(EXIT\_FAILURE);

    } else if (pid == 0) {

        if (input != STDIN\_FILENO) { //dau vao`

            dup2(input, STDIN\_FILENO);

            close(input);

        }

        if (output != STDOUT\_FILENO) { //dau ra

            dup2(output, STDOUT\_FILENO);

            close(output);

        }

        execvp(args[0], args);

        perror("execvp");

        exit(EXIT\_FAILURE);

    } else {

*int* status;

        waitpid(pid, &status, 0);

    }

}

*void* execute\_pipeline(*char* \**args1*[], *char* \**args2*[]) {

*int* pipe\_fd[2];

    if (pipe(pipe\_fd) == -1) {

        perror("pipe");

        exit(EXIT\_FAILURE);

    }

*pid\_t* pid1 = fork();

    if (pid1 == 0) {

        close(pipe\_fd[0]);

        dup2(pipe\_fd[1], STDOUT\_FILENO);

        close(pipe\_fd[1]);

        execvp(args1[0], args1);

        perror("execvp");

        exit(EXIT\_FAILURE);

    } else if (pid1 > 0) {

        waitpid(pid1, NULL, 0);

*pid\_t* pid2 = fork();

        if (pid2 == 0) {

            close(pipe\_fd[1]);

            dup2(pipe\_fd[0], STDIN\_FILENO);

            close(pipe\_fd[0]);

            execvp(args2[0], args2);

            perror("execvp");

            exit(EXIT\_FAILURE);

        } else if (pid2 > 0) {

            close(pipe\_fd[0]);

            close(pipe\_fd[1]);

            waitpid(pid2, NULL, 0);

        } else {

            perror("fork");

            exit(EXIT\_FAILURE);

        }

    } else {

        perror("fork");

        exit(EXIT\_FAILURE);

    }

}

*void* signal\_handler(*int* *sig*) {

    if (sig == SIGINT && running\_pid > 0) {

        kill(running\_pid, SIGINT);

    }

    else{

        printf("\n");

        exit(EXIT\_SUCCESS);

    }

}

*int* main(*void*) {

    if (signal(SIGINT, signal\_handler) == SIG\_ERR) {

        perror("signal");

        exit(EXIT\_FAILURE);

    }

*char* \*args[MAX\_LINE / 2 + 1];

*int* shouldrun = 1;

    while (shouldrun) {

        printf("it007sh>");

        fflush(stdout);

*char* command[MAX\_LINE]; // mang dung` de chua lenh

        fgets(command, MAX\_LINE, stdin);

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

        if (strcmp(command, "HF") == 0)

        {

            print\_history(); // in ra truoc khi luu "HF" vao lich su

            add\_to\_history(command);

            continue;  //sang vong lap tiep theo

        }

        add\_to\_history(command);

*char* \*arg = strtok(command, " \n");

*int* i = 0;

        while (arg != NULL) {

            args[i++] = arg;

            arg = strtok(NULL, " \n");

        }

        args[i] = NULL;

        // 2 bien output va input de dat co` khi co chuyen dau ra/vao ('>' '<')

*int* output = 0;

*int* input = 0;

*int* pipeline = 0;

        for (*int* j = 0; j < i; j++) {

            if (strcmp(args[j], ">") == 0) {

                output = 1;

                args[j] = NULL;

*int* fd = open(args[j + 1], O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

                if (fd == -1) {

                    perror("open");

                    exit(EXIT\_FAILURE);

                }

                execute\_redirection(args, STDIN\_FILENO, fd); // tach thanh` ham` rieng de gon gang hon

                break;

            } else if (strcmp(args[j], "<") == 0) {

                input = 1;

                args[j] = NULL;

*int* fd = open(args[j + 1], O\_RDONLY);

                if (fd == -1) {

                    perror("open");

                    exit(EXIT\_FAILURE);

                }

                execute\_redirection(args, fd, STDOUT\_FILENO);

                break;

            }

            else if (strcmp(args[j], "|") == 0) { // xet pipeline

                pipeline = 1;

                args[j] = NULL;

*char* \*args2[MAX\_LINE / 2 + 1];

*int* k = 0;

                for (*int* l = j + 1; l < i; l++) {

                    args2[k++] = args[l];

                }

                args2[k] = NULL;

                execute\_pipeline(args, args2);

                break;

            }

        }

        // xu li binh thuong khi khong co redirect, pipeline

        if (!output && !input && !pipeline) {

            execute\_command(args);

        }

    }

    return 0;

}

* A screen shot of a computer

  Description automatically generatedA screenshot of a computer screen

  Description automatically generated**Output:**

A black background with white dots

Description automatically generated with medium confidence

* **Giải thích:**
* Gọi signal(SIGINT, signal\_handler) để thiết lập sự kiện xử lý tín hiệu SIGINT (Ctrl+C) bằng cách gọi hàm signal\_handler khi tín hiệu này được nhận.
* Hàm signal\_handler(int sig):
* Điều kiện if (signo == SIGINT) kiểm tra xem tín hiệu nhận được có phải là SIGINT không (Ctrl+C).
* Nếu có tiến trình đang chạy (running\_pid != 0), thì gửi tín hiệu SIGINT đến tiến trình đó bằng hàm kill(running\_pid, SIGINT). Điều này giúp dừng tiến trình đang chạy khi người dùng nhấn Ctrl+C.
* Nếu không có tiến trình nào đang chạy (running\_pid == 0), thoát chương trình shell bằng hàm exit(EXIT\_SUCCESS).
* Giải thích kết quả (gọi những tiến trình chạy liên tục: ‘top’, ‘watch date’, ‘yes’:
* ‘top’: hiển thị thông tin động về các tiến trình đang chạy và sử dụng tài nguyên hệ thống.
* ‘watch’: thực hiện lệnh date mỗi 2 giây và hiển thị kết quả.
* ‘yes’: in ra màn hình một dòng chữ "y" liên tục.