

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thê Tùng.

Họ và tên: Nguyễn Gia Bảo

Mã số sinh viên: 22520109

Lớp: IT007.O29.1

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	<input checked="" type="checkbox"/>				
Chụp hình minh chứng (chạy ít nhất 3 lệnh)	<input checked="" type="checkbox"/>				
Giải thích code, kết quả	<input checked="" type="checkbox"/>				

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

Trả lời...

📝 Ý tưởng thực hiện:

- 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 mảng 2 chiều history để lưu lịch sử lệnh được nhập (tối đa 10).
- 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.

📝 Code thực hiện:

```
#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;
}
```

➡ Kết quả chạy code:

```
it007sh> ls
abc.txt cau1 cau1.c name.txt

it007sh> pwd
/home/ngb/codeLab6

it007sh> cat abc.txt
Hello, I'm 22520109

it007sh> cat name.txt
Nguyen Gia Bao - 22520109

it007sh> ls
abc.txt cau1 cau1.c name.txt

it007sh> HF
Command history:
1. ls
2. pwd
3. cat abc.txt
4. cat name.txt
5. ls

it007sh>
```

➡ Giải thích kết quả:

- Hàm 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():
 - ✓ Tạo một vòng lặp vô hạn để nhận các 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).

Câu 3: Chuyển hướng vào ra

Trả lời...

📝 Ý tưởng thực hiện:

- 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.

✍ Code thực hiện:

```
#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;

        pid = fork();
        if (pid < 0)
            perror("Fork failed");
        else if (pid == 0)
        {
            execvp(cmd, cmd_tokens);
            perror("Execution failed");
            exit(1);
        }
        else
            wait(NULL);
    }
}
```

```
{  
    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);
}
}

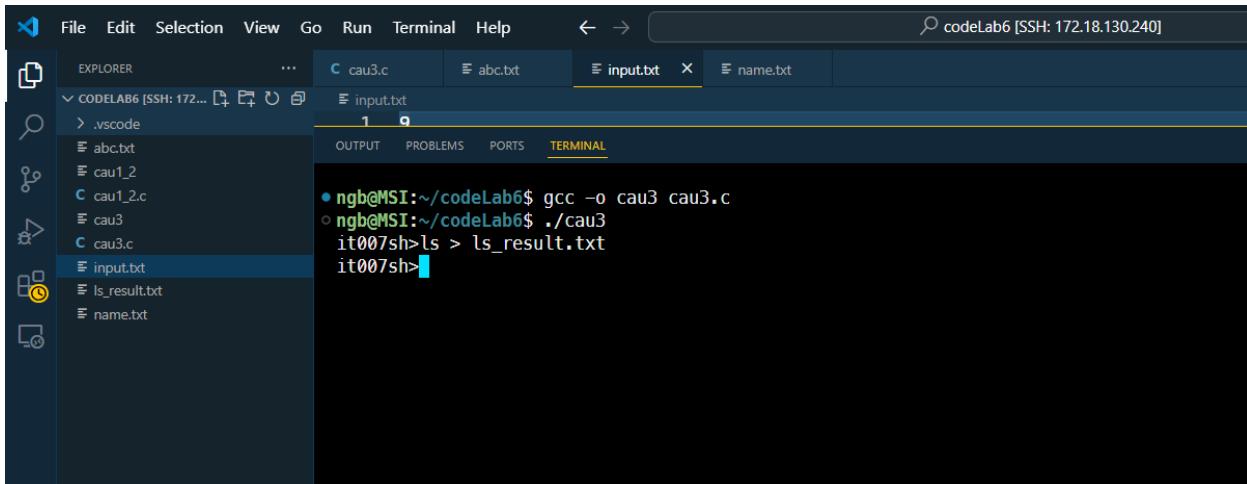
return 0;
}
```



Kết quả chạy code:

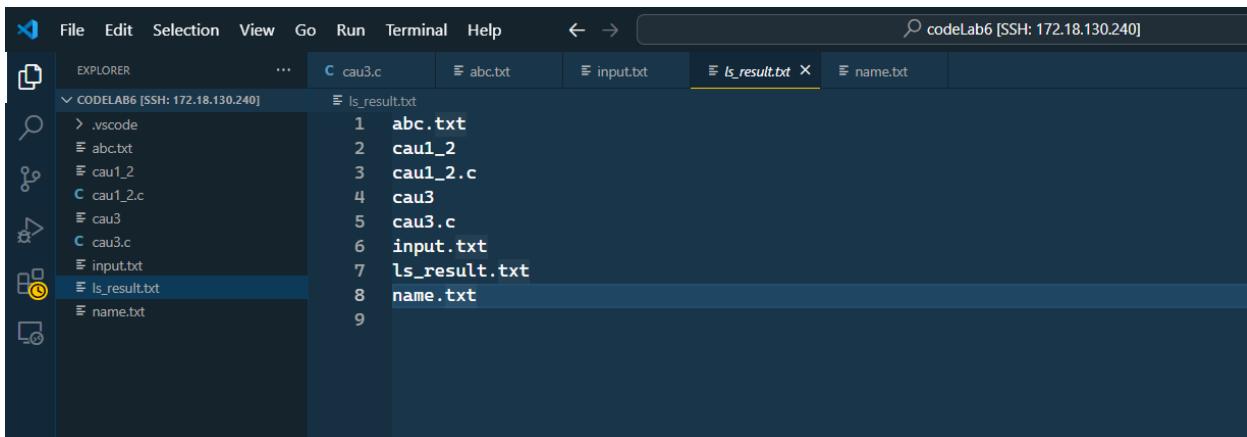
1. Lệnh ls

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thê Tùng.



A screenshot of the Visual Studio Code interface connected via SSH to codeLab6. The terminal tab is active, showing the command `ls` being run. The output shows the directory contents: abc.txt, cau1_2, cau1_2.c, cau3, cau3.c, input.txt, ls_result.txt, and name.txt. The file `ls_result.txt` is currently selected in the Explorer sidebar.

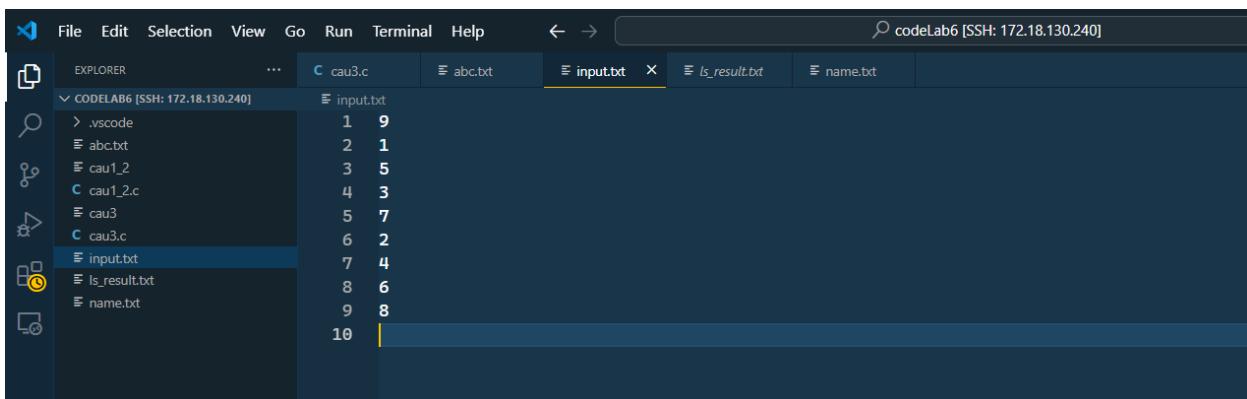
Thực hiện lệnh `ls`, lưu kết quả vào file “`ls_result.txt`”



A screenshot of the Visual Studio Code interface connected via SSH to codeLab6. The terminal tab is active, showing the command `cat ls_result.txt` being run. The output displays the directory contents in a sorted manner: abc.txt, cau1_2, cau1_2.c, cau3, cau3.c, input.txt, ls_result.txt, and name.txt. The file `ls_result.txt` is currently selected in the Explorer sidebar.

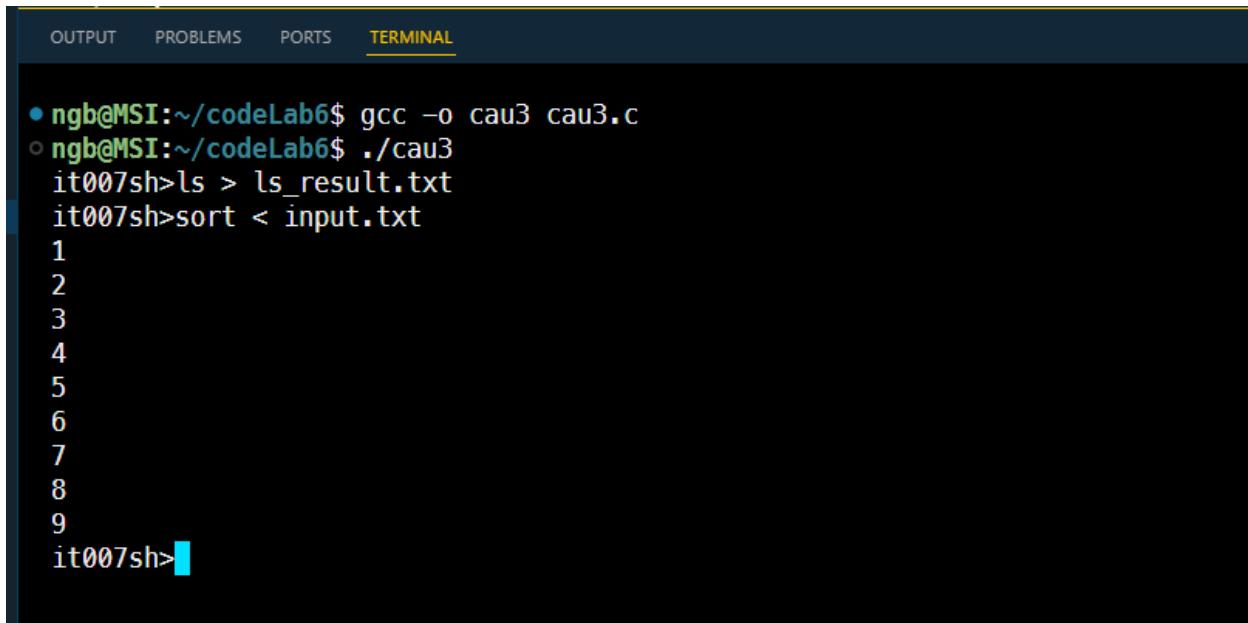
Kết quả: nội dung của file “`ls_result`”

2. Lệnh sort



A screenshot of the Visual Studio Code interface connected via SSH to codeLab6. The terminal tab is active, showing the command `cat input.txt` being run. The output displays the numbers 1 through 10 in a random order: 9, 1, 5, 3, 7, 2, 4, 6, 8, and 10. The file `input.txt` is currently selected in the Explorer sidebar.

File “`input.txt`” chưa được sort

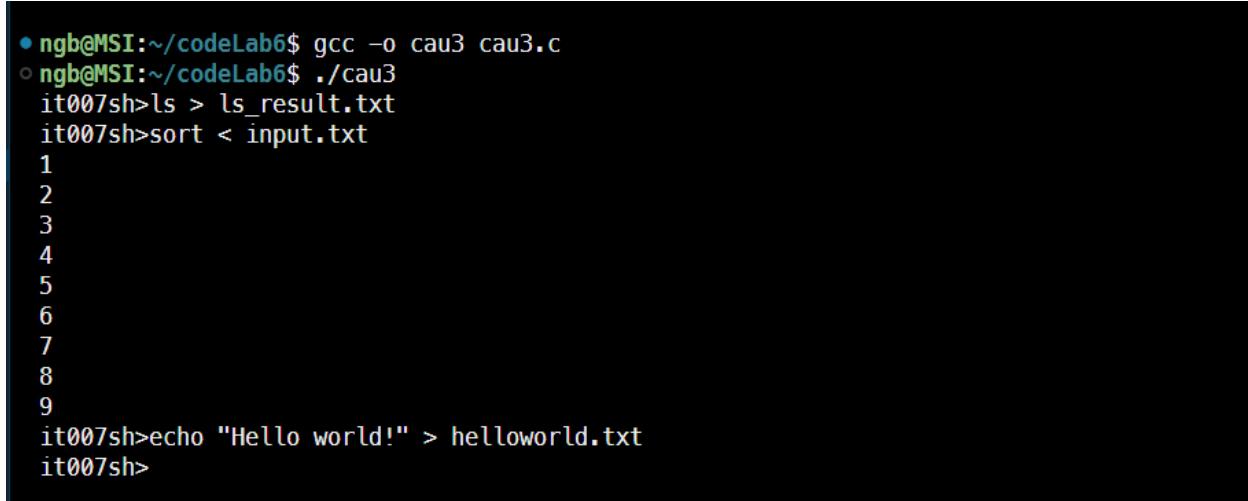


The screenshot shows a terminal window with tabs for OUTPUT, PROBLEMS, PORTS, and TERMINAL. The TERMINAL tab is active, displaying the following command-line session:

```
● ngb@MSI:~/codeLab6$ gcc -o cau3 cau3.c
○ ngb@MSI:~/codeLab6$ ./cau3
it007sh>ls > ls_result.txt
it007sh>sort < input.txt
1
2
3
4
5
6
7
8
9
it007sh>
```

Thực hiện lệnh sort lên file "input.txt"

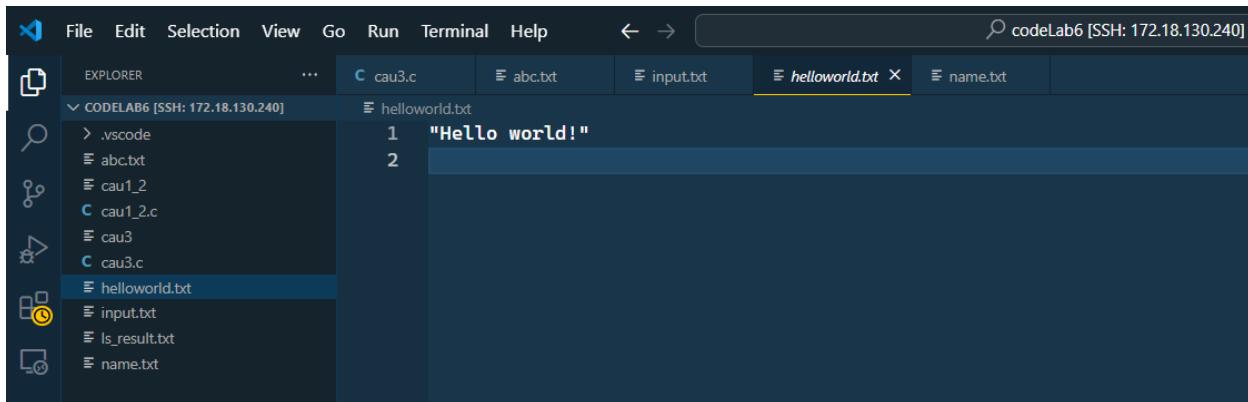
3. Lệnh gán



The screenshot shows a terminal window with the same command-line session as the previous one, followed by an additional command:

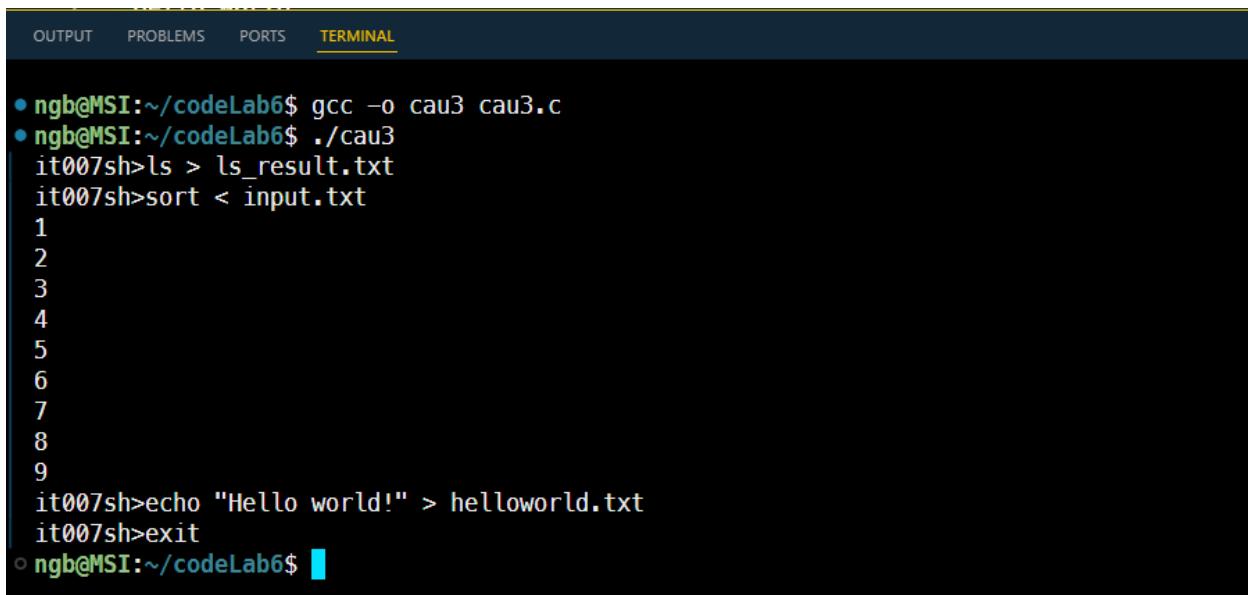
```
● ngb@MSI:~/codeLab6$ gcc -o cau3 cau3.c
○ ngb@MSI:~/codeLab6$ ./cau3
it007sh>ls > ls_result.txt
it007sh>sort < input.txt
1
2
3
4
5
6
7
8
9
it007sh>echo "Hello world!" > helloworld.txt
it007sh>
```

Thực hiện lệnh gán "Hello world" vào file helloworld.txt



The screenshot shows the VS Code interface connected via SSH to codeLab6. The Explorer sidebar shows a folder named 'CODELAB6 [SSH: 172.18.130.240]' containing files like .vscode, abc.txt, cau1_2, cau1_2.c, cau3, and cau3.c. The 'TERMINAL' tab shows the command 'echo "Hello world!" > helloworld.txt' being run, resulting in the file content '1 "Hello world!"' and '2'.

Nội dung của file "helloworld.txt" mới được tạo sau khi thực thi lệnh



The terminal window shows the following session:

```
• ngb@MSI:~/codeLab6$ gcc -o cau3 cau3.c
• ngb@MSI:~/codeLab6$ ./cau3
it007sh>ls > ls_result.txt
it007sh>sort < input.txt
1
2
3
4
5
6
7
8
9
it007sh>echo "Hello world!" > helloworld.txt
it007sh>exit
○ ngb@MSI:~/codeLab6$
```

Nhập "exit" sẽ kết thúc chương trình

➡ **Giải thích kết quả:**

- Chương trình này sẽ hiển thị dấu nhắc it007sh> và chờ người dùng nhập vào một lệnh.
- Nếu người dùng nhập exit, chương trình sẽ kết thúc.
- Nếu không, chương trình sẽ tách lệnh thành các đối số bằng cách sử dụng khoảng trắng làm dấu phân cách.
- 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.
- Chương trình sau đó tạo một tiến trình con bằng cách sử dụng fork().
- 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().

Câu 4: Giao tiếp sử dụng cơ chế đường ống

✍ Ý tưởng thực hiện:

- 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.

✍ 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];

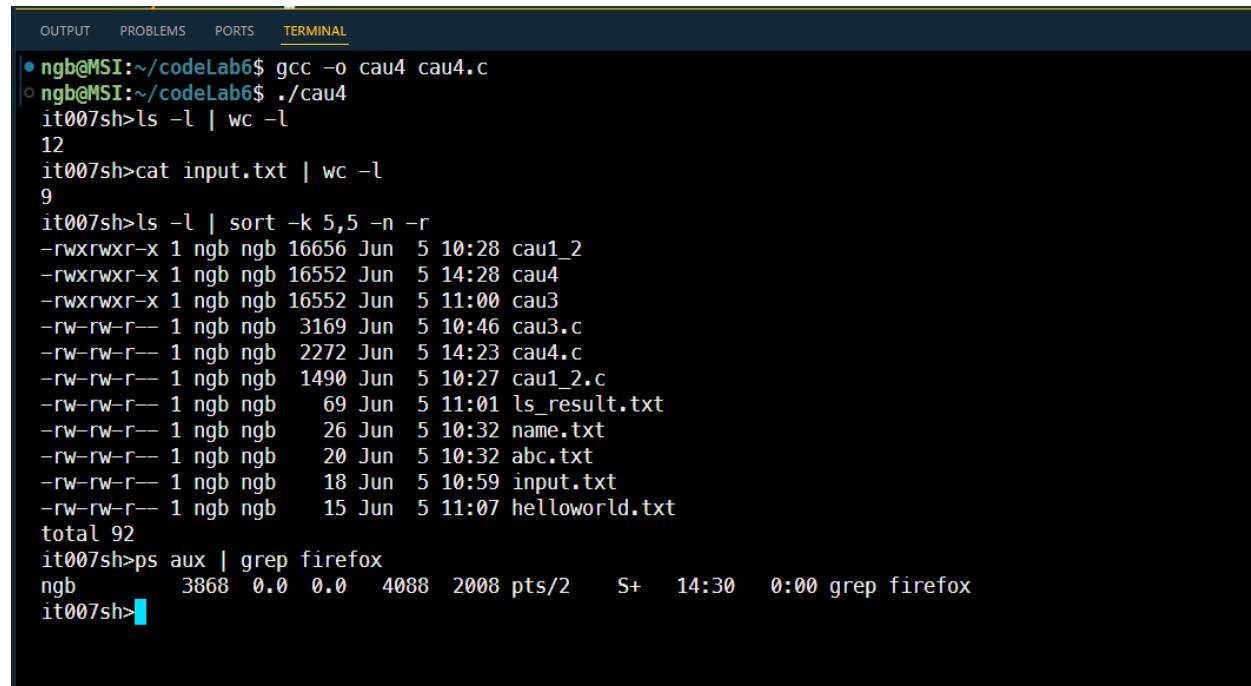
    while (1)
    {
```

```
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;
}
```

➡ Kết quả chạy code:



```
OUTPUT PROBLEMS PORTS TERMINAL
|• ngb@MSI:~/codeLab6$ gcc -o cau4 cau4.c
|◦ ngb@MSI:~/codeLab6$ ./cau4
it007sh>ls -l | wc -l
12
it007sh>cat input.txt | wc -l
9
it007sh>ls -l | sort -k 5,5 -n -r
-rwxrwxr-x 1 ngb ngb 16656 Jun  5 10:28 cau1_2
-rwxrwxr-x 1 ngb ngb 16552 Jun  5 14:28 cau4
-rwxrwxr-x 1 ngb ngb 16552 Jun  5 11:00 cau3
-rw-rw-r-- 1 ngb ngb  3169 Jun  5 10:46 cau3.c
-rw-rw-r-- 1 ngb ngb  2272 Jun  5 14:23 cau4.c
-rw-rw-r-- 1 ngb ngb  1490 Jun  5 10:27 cau1_2.c
-rw-rw-r-- 1 ngb ngb    69 Jun  5 11:01 ls_result.txt
-rw-rw-r-- 1 ngb ngb    26 Jun  5 10:32 name.txt
-rw-rw-r-- 1 ngb ngb    20 Jun  5 10:32 abc.txt
-rw-rw-r-- 1 ngb ngb    18 Jun  5 10:59 input.txt
-rw-rw-r-- 1 ngb ngb    15 Jun  5 11:07 helloworld.txt
total 92
it007sh>ps aux | grep firefox
ngb      3868  0.0  0.0  4088  2008 pts/2    S+   14:30   0:00 grep firefox
it007sh>
```

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 kết quả:

- Chương trình này sẽ hiển thị dấu nhắc `it007sh>` và chờ người dùng nhập vào một lệnh.
- Nếu người dùng nhập `exit`, chương trình sẽ kết thúc.
- Nếu không, chương trình sẽ tách lệnh thành các đối số bằng cách sử dụng khoảng trắng làm dấu phân cách.
- Chương trình sau đó tìm kiếm toán tử `||`. Nếu tìm thấy, nó sẽ ghi nhớ vị trí của nó.
- Chương trình sau đó tạo một tiến trình con bằng cách sử dụng `fork()`.
- Trong tiến trình con:
 - Nếu có toán tử `||`, 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 `||`.
 - 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 `||`.
 - Nếu không có toán tử `||`, nó chỉ đơn giản thực hiện lệnh bằng cách sử dụng `execvp()`.
- Trong tiến trình cha, nó chờ đợi tiến trình con kết thúc bằng cách sử dụng `wait()`.

Câu 5: Kết thúc lệnh đang thực thi

Ý tưởng thực hiện:

- 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 execvp() 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

❖ *Code:*

```
#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;
}
```

➡ Kết quả chạy code:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
634	ngb	20	0	968816	107344	40232	S	0.7	2.9	0:13.50	node
675	ngb	20	0	1014148	70908	37964	S	0.7	1.9	0:15.93	node
3915	ngb	20	0	4271228	23096	13220	S	0.7	0.6	0:00.09	cppTools-srv
768	ngb	20	0	15124	7108	5052	S	0.3	0.2	0:01.78	sshd
788	ngb	20	0	52508	15632	10532	S	0.3	0.4	0:05.28	code-dc96b837cf
833	ngb	20	0	851576	54492	38140	S	0.3	1.4	0:01.67	node
1	root	20	0	21894	13052	9556	S	0.0	0.3	0:00.74	systemd
2	root	20	0	2280	1300	1188	S	0.0	0.0	0:00.00	init-systemd(ub
7	root	20	0	2296	132	132	S	0.0	0.0	0:00.00	init
58	root	20	0	50356	16628	15528	S	0.0	0.4	0:00.42	systemd-journal
73	root	20	0	23972	6044	4888	S	0.0	0.2	0:00.11	systemd-udev
136	systemd+	20	0	21452	11840	9648	S	0.0	0.3	0:00.09	systemd-resolve
137	systemd+	20	0	91020	6560	5704	S	0.0	0.2	0:00.19	systemd-timesyn
174	root	19	-1	18136	8436	7420	S	0.0	0.2	0:00.10	systemd-logind
176	root	20	0	4236	2692	2456	S	0.0	0.1	0:00.02	cron
177	message+	20	0	9620	5248	4560	S	0.0	0.1	0:00.12	dbus-daemon
189	root	20	0	1756096	16060	9476	S	0.0	0.4	0:00.32	wsl-pro-service
239	syslog	20	0	222508	7276	4444	S	0.0	0.2	0:00.13	rsyslogd
244	root	20	0	107000	22268	12936	S	0.0	0.6	0:00.07	unattended-upgr
330	root	20	0	3160	1156	1068	S	0.0	0.0	0:00.00	agetty
333	root	20	0	3116	1144	1060	S	0.0	0.0	0:00.00	agetty
374	root	20	0	2296	112	0	S	0.0	0.0	0:00.00	SessionLeader
375	root	20	0	2296	120	0	S	0.0	0.0	0:00.00	Relay(376)
376	ngb	20	0	6072	5160	3436	S	0.0	0.1	0:00.03	bash
377	root	20	0	6820	4628	3848	S	0.0	0.1	0:00.00	login
462	ngb	20	0	20260	11344	9268	S	0.0	0.3	0:00.05	systemd
463	ngb	20	0	21148	1732	4	S	0.0	0.0	0:00.00	(sd-pam)

Báo cáo thực hành môn Hệ điều hành - Giảng viên: Thân Thê Tùng.

```
OUTPUT PROBLEMS PORTS TERMINAL
377 root    20  0   6820  4628  3848 S  0.0  0.1  0:00.00 login
462 ngb    20  0  20260 11344  9268 S  0.0  0.3  0:00.05 systemd
463 ngb    20  0  21148 1732   4 S  0.0  0.0  0:00.00 (sd-pam)
476 ngb    20  0   6072  5220  3588 S  0.0  0.1  0:00.01 bash
538 root    20  0  12020  7876  6740 S  0.0  0.2  0:00.00 sshd
630 ngb    20  0   2800  1104  1012 S  0.0  0.0  0:00.00 sh
IT007sh> ps
PID TTY      TIME CMD
3545 pts/2  00:00:00 bash
3975 pts/2  00:00:00 cau5
4198 pts/2  00:00:00 ps
IT007sh> df
Filesystem 1K-blocks Used Available Use% Mounted on
none        1880060   4  1880056  1% /mnt/wsl
none        167571452 143263344 24308108 80% /usr/lib/wsl/drivers
none        1880060   0  1880060  0% /usr/lib/modules
none        1880060   0  1880060  0% /usr/lib/modules/5.15.146.1-microsoft-standard-WSL2
/dev/sdc   1055762868 2394060 999665336 1% /
none        1880060   92  1879968  1% /mnt/wslg
none        1880060   0  1880060  0% /usr/lib/wsl/lib
rootfs     1876800  1884  1874916  1% /init
none        1880060   572  1879488  1% /run
none        1880060   0  1880060  0% /run/lock
none        1880060   0  1880060  0% /run/shm
tmpfs      4096    0  4096   0% /sys/fs/cgroup
none        1880060   76  1879984  1% /mnt/wslg/versions.txt
none        1880060   76  1879984  1% /mnt/wslg/doc
C:\       167571452 143263344 24308108 80% /mnt/c
D:\       314571772 186032840 128538932 60% /mnt/d
tmpfs      376012   16  375996  1% /run/user/1002
IT007sh> éit
execvp: No such file or directory
IT007sh> exit
○ ngb@MSI:~/codeLab6$
```

Thực hiện lệnh top và dừng lệnh đang thực thi bằng tổ hợp Ctrl+C và tiếp tục sử dụng terminal cho các lệnh khác

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 dùng nhập tiếp câu lệnh tiếp theo
- 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.