

Name: Đỗ Hoàng Anh

ID: 22520041

Class: IT007.O212.1

## OPERATING SYSTEM LAB 6 REPORT

### SUMMARY

| Task          |  | Status     | Page |
|---------------|--|------------|------|
| Section 6.3.1 |  | Hoàn thành | 2    |
|               |  |            |      |
|               |  |            |      |
|               |  |            |      |
| Section 6.5   |  | Hoàn thành | 4    |

Self-scores: 10/10

*\*Note: Export file to **PDF** and name the file by following format:  
**LAB X – <Student ID>.pdf***

### 6.3.1 Câu hỏi chuẩn bị

Sinh viên chuẩn bị câu trả lời cho câu hỏi sau trước khi bắt đầu phần thực hành:

- 🚦 Hãy nêu các định nghĩa, chức năng và ví dụ sử dụng của các hàm sau đây: `exec()`, `dup2()`, `pipe()`.

Trả lời câu hỏi:

- Hàm `exec()`
  - Định nghĩa:
    - Hàm `exec()` trong ngữ cảnh lập trình C thường liên quan đến các hàm trong họ `exec` trong thư viện chuẩn UNIX (như `execl`, `execp`, `execv`, `execle`, `execve`, `execvp`). Các hàm này thay thế tiến trình hiện tại bằng một chương trình mới được chỉ định.
  - Chức năng:
    - Thay thế hình ảnh của tiến trình hiện tại bằng hình ảnh của một chương trình mới.
    - Không trở lại tiến trình gọi nếu hàm `exec` thành công, nghĩa là chương trình cũ kết thúc và chương trình mới bắt đầu chạy từ đầu.
  - Ví dụ sử dụng:

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main() {
5      char *args[] = {"ls", "-l", NULL};
6      execvp("ls", args);
7
8      // Nếu exec thành công, đoạn code dưới sẽ không bao giờ được thực thi
9      perror("execvp failed");
10     return 1;
11 }
12
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
● anhanh@anhAnh:~/hdh$ ./exec_example
total 24
-rwxr-xr-x 1 anhanh anhanh 17408 Jun 13 13:20 exec_example
-rw-r--r-- 1 anhanh anhanh 261 Jun 13 13:20 exec_example.c
○ anhanh@anhAnh:~/hdh$
```

## – Hàm **dup2()**

### ○ Định nghĩa:

- Hàm **dup2()** sao chép một mô tả tệp (file descriptor) đến một mô tả tệp khác được chỉ định.

### ○ Chức năng:

- Tạo một bản sao của một mô tả tệp hiện có và gán nó vào một mô tả tệp khác.
- Đảm bảo rằng mô tả tệp đích được gán sẽ được đóng trước khi sao chép (nếu nó đã mở).

### ○ Ví dụ:

```
4
5 int main() {
6     int old_fd = open("file.txt", O_RDONLY); // Mở file.txt và gán mô tả tệp cho old_fd
7     int new_fd = 10; // Định nghĩa mô tả tệp mới
8
9     if (dup2(old_fd, new_fd) == -1) {
10         perror("dup2 failed");
11         return 1;
12     }
13
14     // Bây giờ, new_fd có thể được sử dụng để đọc từ file.txt
15     close(old_fd);
16     // Sử dụng new_fd để thực hiện các thao tác I/O
17     // Không tạo tệp mới trên hệ thống, chỉ sao chép mô tả tệp
18     char buffer[100];
19     read(new_fd, buffer, sizeof(buffer)); // Đọc từ new_fd
20     printf("Content: %s\n", buffer);
21
22     close(new_fd); // Đóng mô tả tệp mới
23 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
● anhanh@anhAnh:~/hdh$ ./dup2_example
Content: abcd
○ anhanh@anhAnh:~/hdh$
```

## – Hàm **pipe()**

### ○ Định nghĩa:

- Hàm **pipe()** tạo ra một kênh dẫn (pipe) để truyền dữ liệu giữa các tiến trình.

### ○ Chức năng:

- Tạo ra một kênh dẫn giữa một cặp mô tả tệp: một để đọc và một để ghi.
- Truyền dữ liệu một chiều từ đầu ghi đến đầu đọc.

### ○ Ví dụ sử dụng:

```

7
8   if (pipe(fd) == -1) {
9       perror("pipe failed");
10      return 1;
11  }
12
13  if (fork() == 0) {
14      // Tiến trình con
15      close(fd[0]); // Đóng đầu đọc
16      write(fd[1], "Hello from child", 16);
17      close(fd[1]);
18  } else {
19      // Tiến trình cha
20      close(fd[1]); // Đóng đầu ghi
21      read(fd[0], buffer, 16);
22      printf("Parent read: %s\n", buffer);
23      close(fd[0]);
24  }

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS COMMENTS

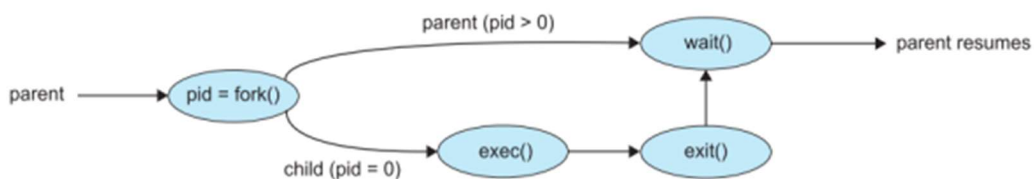
```

● anhanh@anhAnh:~/hdh$ ./pipe_example
Parent read: Hello from child
○ anhanh@anhAnh:~/hdh$

```

## 6.4 Sinh viên thực hành

Hãy thiết kế một chương trình C để tạo ra một giao diện shell. Giao diện này cho phép người dùng nhập các lệnh và sau đó thực thi từng lệnh trong một quy trình riêng biệt. Điểm đặc biệt là chương trình này sẽ hỗ trợ việc chuyển hướng đầu vào và đầu ra, cũng như sử dụng pipes như một cách để truyền thông tin giữa các cặp lệnh. Giao diện shell cung cấp cho người dùng lời nhắc, sau đó lệnh được nhập. Mỗi khi lệnh thực hiện xong, shell sẽ hiện lên dấu nhắc để chạy lệnh khác. Ví dụ dưới đây minh họa dấu nhắc `it007sh>` và lệnh của người dùng: `echo abc` `it007sh> echo abc` Một kỹ thuật để triển khai giao diện shell là trước tiên tiến trình cha đọc những gì người dùng nhập vào dòng lệnh (trong trường hợp này, là lệnh "echo abc"), sau đó tạo ra một tiến trình con riêng biệt để thực hiện lệnh đó. Trừ khi được chỉ định khác, tiến trình cha đợi cho đến khi tiến trình con thoát ra trước khi tiếp tục. Điều này tương tự về chức năng với việc tạo ra một tiến trình mới như được minh họa dưới đây:



Hình 1. Tạo process sử dụng `fork()`

Đoạn chương trình mẫu tạo ra dấu nhắc `it007sh>`:

```
#include <stdio.h>
#include <unistd.h>
#define MAX LINE 80 /* The maximum length command */

int main(void) {
    char *args[MAX LINE/2 + 1]; /* command line arguments */
    int should run = 1; /* flag to determine when to exit
    program */
        while (should run) {
            printf("it007sh>");
            fflush(stdout);
            /**
            Do something
            */
        }
    return 0;
}
```

Dựa vào đoạn chương trình trên hãy thực hiện thêm các yêu cầu dưới đây:

1. Thực thi lệnh trong tiến trình con

🔗 Ví dụ: khi thực hiện

`it007sh> echo abc`

Kết quả sẽ in ra chuỗi `abc`, kết thúc dòng lệnh sẽ hiển thị dấu nhắc `it007sh>` để người dùng nhập lệnh tiếp theo. Lưu ý rằng trong khi lệnh **echo abc** đang thực thi, không cho người dùng nhập command mới.

🔗 Gợi ý: Xem hình 1.

Để thực hiện các lệnh trong tiến trình con và hiển thị kết quả lệnh trong chương trình shell đơn giản `it007sh`, có thể sử dụng các hàm `fork()`, `execvp()`, và `waitpid()`

Trong đó:

**fork()** để tạo tiến trình con.

**execvp()** trong tiến trình con, sử dụng để thực thi lệnh. Nếu **execvp()** thất bại, in thông báo lỗi.

**waitpid()** trong tiến trình cha, sử dụng để đợi tiến trình con kết thúc trước khi tiếp tục nhận lệnh mới.

Chương trình thực thi:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#define MAX_LINE 80 /* The maximum length command */

int main(void) {
    char *args[MAX_LINE/2 + 1]; /* command line arguments */
    int should_run = 1; /* flag to determine when to exit program */

    while (should_run) {
        printf("it007sh> ");
        fflush(stdout);

        char input[MAX_LINE];
        if (!fgets(input, MAX_LINE, stdin)) {
            // Xử lý lỗi khi đọc dòng lệnh
            perror("fgets failed");
            continue;
        }

        // Xóa ký tự newline ở cuối dòng nếu có
        size_t length = strlen(input);
        if (input[length - 1] == '\n') {
            input[length - 1] = '\0';
        }

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

```

// Kiểm tra nếu lệnh là "exit" thì thoát
if (strcmp(args[0], "exit") == 0) {
    should_run = 0;
    continue;
}

pid_t pid = fork();
if (pid < 0) {
    // Xử lý lỗi khi tạo tiến trình con
    perror("Fork failed");
    return 1;
} else if (pid == 0) {
    // Tiến trình con
    if (execvp(args[0], args) == -1) {
        perror("execvp failed");
    }
    return 1;
} else {
    // Tiến trình cha
    waitpid(pid, NULL, 0); // Đợi tiến trình con kết thúc
}

return 0;
}

```

Kết quả chạy thử:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
anhanh@anhAnh:~/hdh$ ./6.4.1
it007sh> ls
6.4.1 6.4.1.c dup2_example dup2_example.c exec_example exec_example.c file.txt pipe_example pipe_example.c
it007sh> ls -l
total 100
-rwxr-xr-x 1 anhanh anhanh 19800 Jun 13 13:50 6.4.1
-rw-r--r-- 1 anhanh anhanh 1721 Jun 13 13:50 6.4.1.c
-rwxr-xr-x 1 anhanh anhanh 17984 Jun 13 13:31 dup2_example
-rw-r--r-- 1 anhanh anhanh 766 Jun 13 13:31 dup2_example.c
-rwxr-xr-x 1 anhanh anhanh 17408 Jun 13 13:20 exec_example
-rw-r--r-- 1 anhanh anhanh 261 Jun 13 13:20 exec_example.c
-rw-r--r-- 1 anhanh anhanh 4 Jun 13 13:29 file.txt
-rwxr-xr-x 1 anhanh anhanh 18072 Jun 13 13:34 pipe_example
-rw-r--r-- 1 anhanh anhanh 546 Jun 13 13:33 pipe_example.c
it007sh> ^C
anhanh@anhAnh:~/hdh$

```

## 2. Tạo tính năng sử dụng lại câu lệnh gần đây

Cho phép người dùng thực thi lệnh gần đây bằng cách sử dụng các phím lên/xuống (để chọn lệnh) và nhấn Enter.

Ví dụ: Nếu các lệnh đã nhập vào shell (theo thứ tự)

echo abc

ls -l

pwd

thì khi sử dụng các phím lên (↑) /xuống (↓), shell sẽ lần lượt hiển thị lại các lệnh trên để người dùng lựa chọn. Nhấn phím Enter để thực thi lệnh đang hiển thị.

Để thêm tính năng sử dụng lại câu lệnh gần đây bằng cách sử dụng các phím lên/xuống, chúng ta cần sử dụng thư viện `termios` và `unistd` để xử lý các sự kiện bàn phím không được bộ đệm.

### Thư viện và biến toàn cục:

**#include <termios.h>:** Thư viện để xử lý các sự kiện bàn phím.

**orig\_termios:** Lưu trữ cài đặt terminal gốc để có thể khôi phục lại khi chương trình kết thúc.

### Chế độ không đệm:

**enableRawMode():** Kích hoạt chế độ không đệm để có thể đọc các ký tự ngay lập tức mà không cần nhấn Enter.

**disableRawMode():** Khôi phục cài đặt terminal gốc khi chương trình kết thúc.

### Lịch sử lệnh:

**history:** Mảng lưu trữ các lệnh đã nhập.

**history\_count:** Đếm số lượng lệnh đã lưu trữ.

**history\_index:** Chỉ số để điều hướng trong lịch sử lệnh khi người dùng nhấn phím lên/xuống.

### Xử lý đầu vào từ người dùng:

Sử dụng **read(STDIN\_FILENO, &c, 1)** để đọc từng ký tự từ đầu vào.

Kiểm tra nếu ký tự là '\n', ESC (phím lên/xuống), hoặc ký tự thông thường để xử lý đầu vào.

### Cập nhật và thực thi lệnh:

Lưu lệnh vào lịch sử.

Tách dòng lệnh thành các tham số và thực thi lệnh bằng **fork()** và **execvp()**.



Nếu lệnh là "exit", thoát khỏi vòng lặp và kết thúc chương trình.

Chương trình hoàn chỉnh:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <sys/wait.h>
5  #include <string.h>
6  #include <termios.h>
7  #include <stdlib.h>
8
9  #define MAX_LINE 80 /* The maximum length command */
10 #define HISTORY_COUNT 10 /* Number of commands to store in history */
11
12 struct termios orig_termios;
13
14 void disableRawMode() {
15     tcsetattr(STDIN_FILENO, TCSAFLUSH, &orig_termios);
16 }
17
18 void enableRawMode() {
19     tcgetattr(STDIN_FILENO, &orig_termios);
20     atexit(disableRawMode);
21
22     struct termios raw = orig_termios;
23     raw.c_lflag &= ~(ECHO | ICANON);
24
25     tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
26 }
27
28 int main(void) {
29     char *args[MAX_LINE/2 + 1]; /* command line arguments */
30     char history[HISTORY_COUNT][MAX_LINE]; /* command history */
31     int history_count = 0; /* number of commands in history */
32     int should_run = 1; /* flag to determine when to exit program */
33     int history_index = -1; /* index for navigating command history */
34 }
```

```

int main(void) {
    enableRawMode();

    while (should_run) {
        printf("it007sh> ");
        fflush(stdout);

        char input[MAX_LINE] = {0};
        int index = 0;

        // Đọc đầu vào từ người dùng
        while (1) {
            char c;
            if (read(STDIN_FILENO, &c, 1) == 1) {
                if (c == '\n') {
                    input[index] = '\0';
                    putchar('\n');
                    break;
                } else if (c == 27) {
                    char seq[3];
                    if (read(STDIN_FILENO, &seq[0], 1) == 1 && read(STDIN_FILENO, &seq[1], 1) == 1) {
                        if (seq[0] == '[') {
                            if (seq[1] == 'A') {
                                // Phím lên
                                if (history_index < history_count - 1) {
                                    history_index++;
                                    strcpy(input, history[history_count - history_index - 1]);
                                    index = strlen(input);
                                    printf("\33[2K\r"); // Xóa dòng hiện tại
                                    printf("it007sh> %s", input);
                                    fflush(stdout);
                                }
                            } else if (seq[1] == 'B') {
                                // Phím xuống
                                if (history_index > 0) {
                                    history_index--;
                                    strcpy(input, history[history_count - history_index - 1]);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

int main(void) {
    printf("\33[2K\r"); // Xóa dòng hiện tại
    printf("it007sh> %s", input);
    fflush(stdout);
} else if (history_index == 0) {
    history_index--;
    input[0] = '\0';
    index = 0;
    printf("\33[2K\r"); // Xóa dòng hiện tại
    printf("it007sh> ");
    fflush(stdout);
}
}
} else {
    if (index < MAX_LINE - 1) {
        input[index++] = c;
        putchar(c);
        fflush(stdout);
    }
}
}

// Xử lý Lệnh
if (strlen(input) > 0) {
    // Lưu lệnh vào lịch sử
    if (history_count < HISTORY_COUNT) {
        strcpy(history[history_count++], input);
    } else {
        for (int i = 1; i < HISTORY_COUNT; i++) {
            strcpy(history[i - 1], history[i]);
        }
        strcpy(history[HISTORY_COUNT - 1], input);
    }
    history_index = -1; // Reset history index
}
}

```

```

// Tách dòng lệnh thành các tham số
int i = 0;
char *token = strtok(input, " ");
while (token != NULL) {
    args[i++] = token;
    token = strtok(NULL, " ");
}
args[i] = NULL;

// Kiểm tra nếu lệnh là "exit" thì thoát
if (strcmp(args[0], "exit") == 0) {
    should_run = 0;
    continue;
}

pid_t pid = fork();
if (pid < 0) {
    perror("Fork failed");
    return 1;
} else if (pid == 0) {
    if (execvp(args[0], args) == -1) {
        perror("execvp failed");
    }
    return 1;
} else {
    waitpid(pid, NULL, 0); // Đợi tiến trình con kết thúc
}
}

return 0;
}

```

### 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

thì 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

thì tệp in.txt sẽ đóng vai trò là đầu vào cho lệnh sắp xếp.

🔗 Gợi ý: sử dụng hàm dup2()

Để thực thi các tính năng trên ta cần thêm các giá trị:

Bao gồm tệp tiêu đề <sys/stat.h>: Tệp tiêu đề này định nghĩa các macro cho quyền truy cập tệp như S\_IRUSR, S\_IWUSR.

Thiết lập quyền tệp cho đầu ra: Khi mở tệp cho đầu ra (output\_file), sử dụng các quyền S\_IRUSR | S\_IWUSR để chỉ định rằng tệp được tạo có quyền đọc và ghi cho người dùng sở hữu tệp.

Kiểm tra lỗi khi mở tệp: Kiểm tra và báo lỗi nếu không thể mở tệp đầu vào hoặc đầu ra.

Sử dụng dup2() để chuyển hướng: Chuyển hướng đầu vào (STDIN\_FILENO) từ tệp đầu vào và đầu ra (STDOUT\_FILENO) đến tệp đầu ra.

Thực thi lệnh với execvp(): Sau khi thiết lập chuyển hướng, thực thi lệnh bằng execvp(). Nếu thực thi thất bại, báo lỗi bằng perror().

Chương trình hoàn chỉnh

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <termios.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h> // For file permission macros

#define MAX_LINE 80 /* The maximum length command */
#define HISTORY_COUNT 10 /* Number of commands to store in history */

struct termios orig_termios;

void disableRawMode() {
    tcsetattr(STDIN_FILENO, TCSAFLUSH, &orig_termios);
}

void enableRawMode() {
    tcgetattr(STDIN_FILENO, &orig_termios);
    atexit(disableRawMode);

    struct termios raw = orig_termios;
    raw.c_lflag &= ~(ECHO | ICANON);

    tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}

void parse_command(char *input, char **args, char **input_file, char **output_file) {
    char *token;
    int i = 0;

    *input_file = NULL;
    *output_file = NULL;
```

```

// Đọc đầu vào từ người dùng
while (1) {
    char c;
    if (read(STDIN_FILENO, &c, 1) == 1) {
        if (c == '\n') {
            input[index] = '\0';
            putchar('\n');
            break;
        } else if (c == 27) {
            char seq[3];
            if (read(STDIN_FILENO, &seq[0], 1) == 1 && read(STDIN_FILENO, &seq[1], 1) == 1) {
                if (seq[0] == '[') {
                    if (seq[1] == 'A') {
                        // Phím Lên
                        if (history_index < history_count - 1) {
                            history_index++;
                            strcpy(input, history[history_count - history_index - 1]);
                            index = strlen(input);
                            printf("\33[2K\r"); // Xóa dòng hiện tại
                            printf("it007sh> %s", input);
                            fflush(stdout);
                        }
                    } else if (seq[1] == 'B') {
                        // Phím xuống
                        if (history_index > 0) {
                            history_index--;
                            strcpy(input, history[history_count - history_index - 1]);
                            index = strlen(input);
                            printf("\33[2K\r"); // Xóa dòng hiện tại
                            printf("it007sh> %s", input);
                            fflush(stdout);
                        } else if (history_index == 0) {
                            history_index--;
                            input[0] = '\0';
                            index = 0;
                        }
                    }
                }
            }
        }
    }
}

void parse_command(char *input, char **args, char **input_file, char **output_file) {
    token = strtok(input, " ");
    while (token != NULL) {
        if (strcmp(token, "<") == 0) {
            // Xử lý toán tử chuyển hướng đầu vào
            token = strtok(NULL, " ");
            *input_file = token;
        } else if (strcmp(token, ">") == 0) {
            // Xử lý toán tử chuyển hướng đầu ra
            token = strtok(NULL, " ");
            *output_file = token;
        } else {
            // Lưu tham số vào args
            args[i++] = token;
        }
        token = strtok(NULL, " ");
    }
    args[i] = NULL;
}

int main(void) {
    char *args[MAX_LINE/2 + 1]; /* command line arguments */
    char history[HISTORY_COUNT][MAX_LINE]; /* command history */
    int history_count = 0; /* number of commands in history */
    int should_run = 1; /* flag to determine when to exit program */
    int history_index = -1; /* index for navigating command history */

    enableRawMode();

    while (should_run) {
        printf("it007sh> ");
        fflush(stdout);

        char input[MAX_LINE] = {0};
        int index = 0;

```

```

        printf("\33[2K\r"); // Xóa dòng hiện tại
        printf("it007sh> ");
        fflush(stdout);
    }
}

} else {
    if (index < MAX_LINE - 1) {
        input[index++] = c;
        putchar(c);
        fflush(stdout);
    }
}
}

// Xử lý lệnh
if (strlen(input) > 0) {
    // Lưu lệnh vào lịch sử
    if (history_count < HISTORY_COUNT) {
        strcpy(history[history_count++], input);
    } else {
        for (int i = 1; i < HISTORY_COUNT; i++) {
            strcpy(history[i - 1], history[i]);
        }
        strcpy(history[HISTORY_COUNT - 1], input);
    }
    history_index = -1; // Reset history index

    // Tách dòng lệnh thành các tham số và tìm các tệp chuyển hướng
    char *input_file = NULL;
    char *output_file = NULL;
    parse_command(input, args, &input_file, &output_file);
}

```

```

// Kiểm tra nếu lệnh là "exit" thì thoát
if (args[0] && strcmp(args[0], "exit") == 0) {
    should_run = 0;
    continue;
}

pid_t pid = fork();
if (pid < 0) {
    perror("Fork failed");
    return 1;
} else if (pid == 0) {
    // Chuyển hướng đầu vào
    if (input_file) {
        int fd = open(input_file, O_RDONLY);
        if (fd == -1) {
            perror("open input file");
            return 1;
        }
        dup2(fd, STDIN_FILENO);
        close(fd);
    }

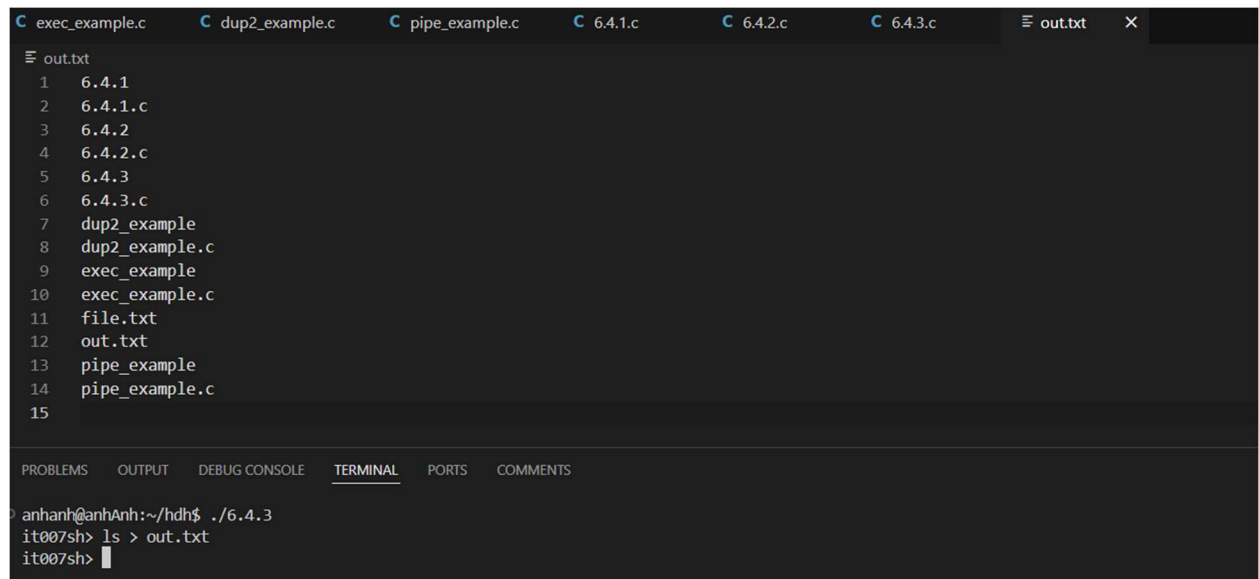
    // Chuyển hướng đầu ra
    if (output_file) {
        int fd = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
        if (fd == -1) {
            perror("open output file");
            return 1;
        }
        dup2(fd, STDOUT_FILENO);
        close(fd);
    }

    if (execvp(args[0], args) == -1) {
        perror("execvp failed");
    }
}

```



Ví dụ chạy thử:



```
exec_example.c  dup2_example.c  pipe_example.c  6.4.1.c  6.4.2.c  6.4.3.c  out.txt X
out.txt
1 6.4.1
2 6.4.1.c
3 6.4.2
4 6.4.2.c
5 6.4.3
6 6.4.3.c
7 dup2_example
8 dup2_example.c
9 exec_example
10 exec_example.c
11 file.txt
12 out.txt
13 pipe_example
14 pipe_example.c
15
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
anhnh@anhAnh:~/hdh$ ./6.4.3
it007sh> ls > out.txt
it007sh>
```

#### 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: Khi người dùng nhập

```
it007sh>ls -l | less
```

thì đầu ra của lệnh `ls -l` đóng vai trò là đầu vào cho lệnh `less`.

Gợi ý: sử dụng hàm `pipe()` và `dup2()`.

Để hỗ trợ giao tiếp sử dụng cơ chế đường ống trong shell đơn giản của bạn, bạn cần sử dụng các hàm `pipe()` và `dup2()` để thiết lập và quản lý đường ống giữa các tiến trình.

Hàm **`parse_command()`** không cần thay đổi:

Hàm **`parse_command()`** vẫn giữ nguyên để tách dòng lệnh thành các đối số (arguments).

Thêm hỗ trợ cho dấu `|` (đường ống):

Chương trình kiểm tra xem có dấu `|` trong lệnh không (`pipe_index`). Nếu có, chia lệnh thành hai phần trước và sau dấu `|`.

Sử dụng hàm **`pipe()`** và **`dup2()`** để thiết lập đường ống:

Nếu có dấu `|`, chương trình tạo một đường ống (`pipefd`) và tạo hai tiến trình con. Tiến trình con thứ nhất thực thi lệnh trước dấu `|`, và đầu ra của nó được đưa vào đầu vào của đường ống. Tiến trình con thứ hai thực thi lệnh sau dấu `|`, và đầu vào của nó được lấy từ đầu ra của đường ống.

## Chương trình hoàn chỉnh:

```
#define MAX_LINE 80 /* The maximum length command */
#define HISTORY_COUNT 10 /* Number of commands to store in history */

struct termios orig_termios;

void disableRawMode() {
    tcsetattr(STDIN_FILENO, TCSAFLUSH, &orig_termios);
}

void enableRawMode() {
    tcgetattr(STDIN_FILENO, &orig_termios);
    atexit(disableRawMode);

    struct termios raw = orig_termios;
    raw.c_lflag &= ~(ECHO | ICANON);

    tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}

void parse_command(char *input, char **args) {
    char *token;
    int i = 0;

    token = strtok(input, " ");
    while (token != NULL) {
        args[i++] = token;
        token = strtok(NULL, " ");
    }
    args[i] = NULL;
}

int main(void) {
    char *args[MAX_LINE/2 + 1]; /* command line arguments */
    char history[HISTORY_COUNT][MAX_LINE]; /* command history */
    int history_count = 0; /* number of commands in history */
    int should_run = 1; /* flag to determine when to exit program */
    int history_index = -1; /* index for navigating command history */
```

```
    enableRawMode();

    while (should_run) {
        printf("it007sh> ");
        fflush(stdout);

        char input[MAX_LINE] = {0};
        int index = 0;

        // Đọc đầu vào từ người dùng
        while (1) {
            char c;
            if (read(STDIN_FILENO, &c, 1) == 1) {
                if (c == '\n') {
                    input[index] = '\0';
                    putchar('\n');
                    break;
                } else if (c == 27) {
                    char seq[3];
                    if (read(STDIN_FILENO, &seq[0], 1) == 1 && read(STDIN_FILENO, &seq[1], 1) == 1) {
                        if (seq[0] == '[') {
                            if (seq[1] == 'A') {
                                // Phím Lên
                                if (history_index < history_count - 1) {
                                    history_index++;
                                    strcpy(input, history[history_count - history_index - 1]);
                                    index = strlen(input);
                                    printf("\33[2K\r"); // Xóa dòng hiện tại
                                    printf("it007sh> %s", input);
                                    fflush(stdout);
                                }
                            } else if (seq[1] == 'B') {
                                // Phím xuống
                                if (history_index > 0) {
                                    history_index--;
                                    strcpy(input, history[history_count - history_index - 1]);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```



```

        index = strlen(input);
        printf("\33[2K\r"); // Xóa dòng hiện tại
        printf("it007sh> %s", input);
        fflush(stdout);
    } else if (history_index == 0) {
        history_index--;
        input[0] = '\0';
        index = 0;
        printf("\33[2K\r"); // Xóa dòng hiện tại
        printf("it007sh> ");
        fflush(stdout);
    }
}
} else {
    if (index < MAX_LINE - 1) {
        input[index++] = c;
        putchar(c);
        fflush(stdout);
    }
}
}
}

```

```

// Xử lý lệnh
if (strlen(input) > 0) {
    // Lưu lệnh vào lịch sử
    if (history_count < HISTORY_COUNT) {
        strcpy(history[history_count++], input);
    } else {
        for (int i = 1; i < HISTORY_COUNT; i++) {
            strcpy(history[i - 1], history[i]);
        }
        strcpy(history[HISTORY_COUNT - 1], input);
    }
}

```

```

history_index = -1; // Reset history index

// Tách dòng lệnh thành các tham số
parse_command(input, args);

// Kiểm tra nếu lệnh là "exit" thì thoát
if (strcmp(args[0], "exit") == 0) {
    should_run = 0;
    continue;
}

// Kiểm tra xem có dấu '|' trong lệnh hay không
int pipe_index = -1;
for (int i = 0; args[i] != NULL; i++) {
    if (strcmp(args[i], "|") == 0) {
        pipe_index = i;
        break;
    }
}

if (pipe_index != -1) {
    // Tạo pipe
    int pipefd[2];
    if (pipe(pipefd) == -1) {
        perror("pipe");
        return 1;
    }

    // Tạo tiến trình con 1
    pid_t pid1 = fork();
    if (pid1 < 0) {
        perror("fork");
        return 1;
    } else if (pid1 == 0) {
        // Thiết lập đầu ra của tiến trình con 1 là đầu vào của pipe
    }
}

```

```

} else if (pid1 == 0) {
    // Thiết lập đầu ra của tiến trình con 1 là đầu vào của pipe
    close(pipefd[0]); // Đóng đầu đọc của pipe
    dup2(pipefd[1], STDOUT_FILENO); // Thay đổi STDOUT thành đầu ra của pipe
    close(pipefd[1]); // Đóng đầu ghi của pipe

    // Thực thi lệnh trước dấu '|'
    args[pipe_index] = NULL; // Kết thúc danh sách tham số ở đây
    if (execvp(args[0], args) == -1) {
        perror("execvp");
        return 1;
    }
} else {
    // Tiến trình cha đợi tiến trình con 1 kết thúc
    waitpid(pid1, NULL, 0);

    // Tạo tiến trình con 2
    pid_t pid2 = fork();
    if (pid2 < 0) {
        perror("fork");
        return 1;
    } else if (pid2 == 0) {
        // Thiết lập đầu vào của tiến trình con 2 là đầu ra của pipe
        close(pipefd[1]); // Đóng đầu ghi của pipe
        dup2(pipefd[0], STDIN_FILENO); // Thay đổi STDIN thành đầu vào của pipe
        close(pipefd[0]); // Đóng đầu đọc của pipe

        // Thực thi lệnh sau dấu '|'
        if (execvp(args[pipe_index + 1], &args[pipe_index + 1]) == -1) {
            perror("execvp");
            return 1;
        }
    } else {
        // Tiến trình cha đợi tiến trình con 2 kết thúc
        close(pipefd[0]);
        close(pipefd[1]);
    }
} else {
    // Không có dấu '|' trong lệnh, chỉ thực thi lệnh đơn
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
        return 1;
    } else if (pid == 0) {
        // Thực thi lệnh
        if (execvp(args[0], args) == -1) {
            perror("execvp");
            return 1;
        }
    } else {
        // Tiến trình cha đợi tiến trình con kết thúc
        waitpid(pid, NULL, 0);
    }
}

return 0;
}

```

Ví dụ chạy thử lệnh `ls -l | less`:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

total 200
-rwxr-xr-x 1 anhanh anhanh 19800 Jun 13 13:50 6.4.1
-rw-r--r-- 1 anhanh anhanh 1721 Jun 13 13:59 6.4.1.c
-rwxr-xr-x 1 anhanh anhanh 21488 Jun 13 14:07 6.4.2
-rw-r--r-- 1 anhanh anhanh 4985 Jun 13 14:07 6.4.2.c
-rwxr-xr-x 1 anhanh anhanh 22152 Jun 13 14:28 6.4.3
-rw-r--r-- 1 anhanh anhanh 6622 Jun 13 14:27 6.4.3.c
-rwxr-xr-x 1 anhanh anhanh 22240 Jun 13 14:38 6.4.4
-rw-r--r-- 1 anhanh anhanh 7958 Jun 13 14:38 6.4.4.c
-rwxr-xr-x 1 anhanh anhanh 17984 Jun 13 13:31 dup2_example
-rw-r--r-- 1 anhanh anhanh 766 Jun 13 13:31 dup2_example.c
-rwxr-xr-x 1 anhanh anhanh 17408 Jun 13 13:20 exec_example
-rw-r--r-- 1 anhanh anhanh 261 Jun 13 13:20 exec_example.c
-rw-r--r-- 1 anhanh anhanh 4 Jun 13 13:29 file.txt
-rw----- 1 anhanh anhanh 143 Jun 13 14:35 out.txt
-rwxr-xr-x 1 anhanh anhanh 18072 Jun 13 13:34 pipe_example
-rw-r--r-- 1 anhanh anhanh 546 Jun 13 13:33 pipe_example.c
~
~
(END)
```

5. Kết thúc lệnh đang thực thi bằng tổ hợp phím `Ctrl + C`

🌈 Ví dụ: Thực hiện lệnh

```
it007sh>top
```

sẽ liên tục hiển thị các tiến trình của hệ thống. Khi đó, nếu 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.

Để hỗ trợ việc kết thúc lệnh đang thực thi bằng tổ hợp phím `Ctrl + C` trong chương trình shell của bạn, bạn cần xử lý tín hiệu `SIGINT` (Interrupt Signal) mà hệ điều hành gửi khi người dùng nhấn tổ hợp phím này. Khi nhận được tín hiệu này, shell cần phải dừng tiến trình hiện tại đang chạy và hiển thị lại dấu nhắc để người dùng nhập lệnh tiếp theo.

Để cải tiến chương trình shell để xử lý tín hiệu **SIGINT**:

Thêm xử lý tín hiệu `SIGINT`:

Trước hết, bạn cần sử dụng hàm `signal()` hoặc `sigaction()` để thiết lập xử lý cho tín hiệu `SIGINT`.

Khi nhận được tín hiệu `SIGINT`, shell sẽ cần ngừng tiến trình con hiện tại (nếu có) và in lại dấu nhắc `it007sh>`.

Cài đặt xử lý tín hiệu:

Trong vòng lặp chính của shell, sử dụng hàm `signal(SIGINT, handler_function)` hoặc `sigaction(SIGINT, &sa, NULL)` để đăng ký hàm xử lý tín hiệu.

Trong hàm xử lý tín hiệu, bạn cần kiểm tra xem có đang có tiến trình con đang chạy hay không, và nếu có, sử dụng kill() để gửi tín hiệu SIGINT đến tiến trình đó.

Chương trình hoàn chỉnh:

```
struct termios orig_termios;

void disableRawMode() {
    tcsetattr(STDIN_FILENO, TCSAFLUSH, &orig_termios);
}

void enableRawMode() {
    tcgetattr(STDIN_FILENO, &orig_termios);
    atexit(disableRawMode);

    struct termios raw = orig_termios;
    raw.c_lflag &= ~(ECHO | ICANON);

    tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}

void parse_command(char *input, char **args) {
    char *token;
    int i = 0;

    token = strtok(input, " ");
    while (token != NULL) {
        args[i++] = token;
        token = strtok(NULL, " ");
    }
    args[i] = NULL;
}

void handle_sigint(int sig) {
    printf("\n");
    fflush(stdout);
}

int main(void) {
    char *args[MAX_LINE/2 + 1]; /* command line arguments */
    char history[HISTORY_COUNT][MAX_LINE]; /* command history */
    int history_count = 0; /* number of commands in history */
    int should_run = 1; /* flag to determine when to exit program */
    int history_index = -1; /* index for navigating command history */

    enableRawMode();
    signal(SIGINT, handle_sigint); // Đăng ký xử lý tín hiệu SIGINT

    while (should_run) {
        printf("it007sh> ");
        fflush(stdout);

        char input[MAX_LINE] = {0};
        int index = 0;

        // Đọc đầu vào từ người dùng
        while (1) {
            char c;
            if (read(STDIN_FILENO, &c, 1) == 1) {
                if (c == '\n') {
                    input[index] = '\0';
                    putchar('\n');
                    break;
                } else if (c == 27) {
                    char seq[3];
                    if (read(STDIN_FILENO, &seq[0], 1) == 1 && read(STDIN_FILENO, &seq[1], 1) == 1) {
                        if (seq[0] == '[') {
                            if (seq[1] == 'A') {
                                // Phím Lên
                                if (history_index < history_count - 1) {
                                    history_index++;
                                    strcpy(input, history[history_count - history_index - 1]);
                                    index = strlen(input);
                                    printf("\33[2K\r"); // Xóa dòng hiện tại
                                }
                            }
                        }
                    }
                }
                input[index++] = c;
            }
        }

        parse_command(input, args);
        if (args[0] != NULL) {
            history_count++;
            if (history_count == HISTORY_COUNT) {
                history_count--;
            }
            strcpy(history[history_count - 1], input);
            history_index = history_count - 1;
        }

        if (args[0] != NULL) {
            printf("%s\n", args[0]);
        }
    }
}
```

```

        fflush(stdout);
    }
} else if (seq[1] == 'B') {
    // Phím xuống
    if (history_index < HISTORY_COUNT) {
        char history[10][80];
        strcpy(history[history_index], input);
        history_index++;
        index = strlen(input);
        printf("\33[2K\r"); // Xóa dòng hiện tại
        printf("it007sh> %s", input);
        fflush(stdout);
    } else if (history_index == 0) {
        history_index--;
        input[0] = '\0';
        index = 0;
        printf("\33[2K\r"); // Xóa dòng hiện tại
        printf("it007sh> ");
        fflush(stdout);
    }
}

} else {
    if (index < MAX_LINE - 1) {
        input[index++] = c;
        putchar(c);
        fflush(stdout);
    }
}
}

// Xử lý lệnh
if (strlen(input) > 0) {
    // Lưu lệnh vào lịch sử
    if (history_count < HISTORY_COUNT) {
        strcpy(history[history_count++], input);
    } else {
        for (int i = 1; i < HISTORY_COUNT; i++) {
            strcpy(history[i - 1], history[i]);
        }
        strcpy(history[HISTORY_COUNT - 1], input);
    }
    history_index = -1; // Reset history index

    // Tách dòng lệnh thành các tham số
    parse_command(input, args);

    // Kiểm tra nếu lệnh là "exit" thì thoát
    if (strcmp(args[0], "exit") == 0) {
        should_run = 0;
        continue;
    }

    // Kiểm tra xem có dấu '|' trong lệnh hay không
    int pipe_index = -1;
    for (int i = 0; args[i] != NULL; i++) {
        if (strcmp(args[i], "|") == 0) {
            pipe_index = i;
            break;
        }
    }

    if (pipe_index != -1) {
        // Tạo pipe
        int pipefd[2];
        if (pipe(pipefd) == -1) {
            perror("pipe");
            return 1;
        }
    }
}

```

```

int main(void) {
    // Tạo tiến trình con 1
    pid_t pid1 = fork();
    if (pid1 < 0) {
        perror("fork");
        return 1;
    } else if (pid1 == 0) {
        // Thiết lập đầu ra của tiến trình con 1 là đầu vào của pipe
        close(pipefd[0]); // Đóng đầu đọc của pipe
        dup2(pipefd[1], STDOUT_FILENO); // Thay đổi STDOUT thành đầu ra của pipe
        close(pipefd[1]); // Đóng đầu ghi của pipe

        // Thực thi lệnh trước dấu '|'
        args[pipe_index] = NULL; // Kết thúc danh sách tham số ở đây
        if (execvp(args[0], args) == -1) {
            perror("execvp");
            return 1;
        }
    } else {
        // Tiến trình cha đợi tiến trình con 1 kết thúc
        waitpid(pid1, NULL, 0);

        // Tạo tiến trình con 2
        pid_t pid2 = fork();
        if (pid2 < 0) {
            perror("fork");
            return 1;
        } else if (pid2 == 0) {
            // Thiết lập đầu vào của tiến trình con 2 là đầu ra của pipe
            close(pipefd[1]); // Đóng đầu ghi của pipe
            dup2(pipefd[0], STDIN_FILENO); // Thay đổi STDIN thành đầu vào của pipe
            close(pipefd[0]); // Đóng đầu đọc của pipe

            // Thực thi lệnh sau dấu '|'
            if (execvp(args[pipe_index + 1], &args[pipe_index + 1]) == -1) {
                return 1;
            }
        } else {
            // Tiến trình cha đóng pipe và đợi tiến trình con 2 kết thúc
            close(pipefd[0]);
            close(pipefd[1]);

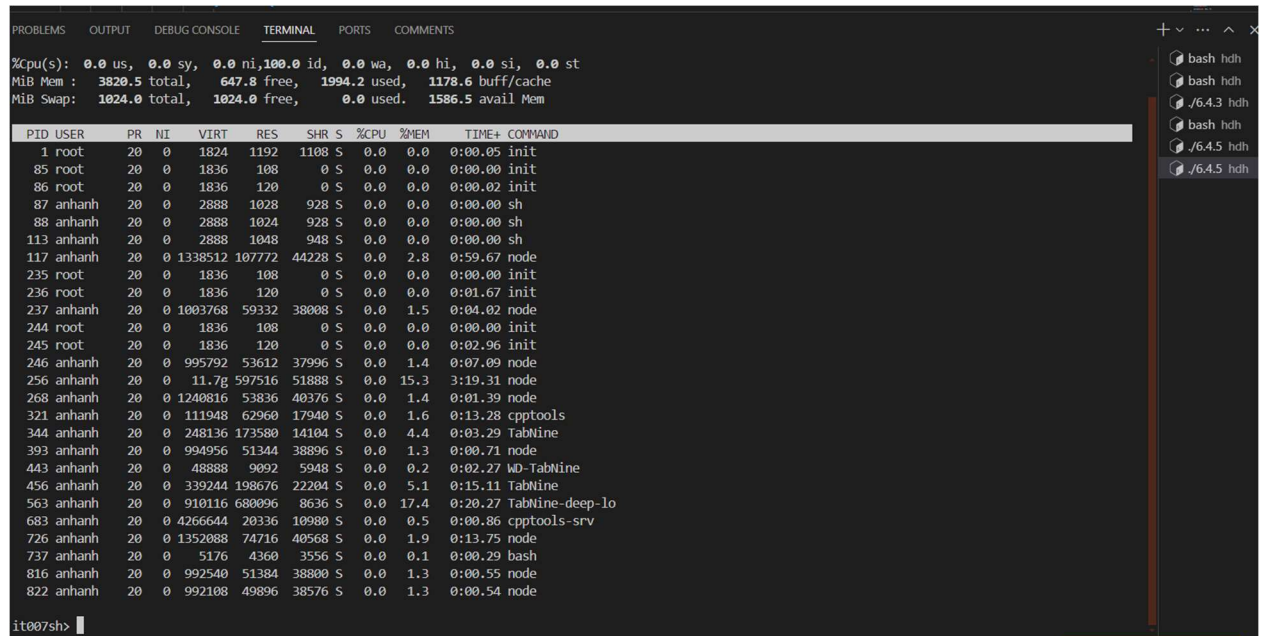
            waitpid(pid2, NULL, 0);
        }
    }
} else {
    // Không có dấu '|' trong lệnh, thực thi lệnh bình thường
    pid_t pid = fork();
    if (pid < 0) {
        perror("fork");
        return 1;
    } else if (pid == 0) {
        // Tiến trình con
        if (execvp(args[0], args) == -1) {
            perror("execvp");
            return 1;
        }
    } else {
        // Tiến trình cha đợi tiến trình con kết thúc
        waitpid(pid, NULL, 0);
    }
}

return 0;
}

```



Khi chạy thử:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
%cpu(s): 0.0 us, 0.0 sy, 0.0 ni,100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3820.5 total, 647.8 free, 1994.2 used, 1178.6 buff/cache
MiB Swap: 1024.0 total, 1024.0 free, 0.0 used, 1586.5 avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 20 0 1824 1192 1108 S 0.0 0.0 0:00.05 init
85 root 20 0 1836 108 0 S 0.0 0.0 0:00.00 init
86 root 20 0 1836 120 0 S 0.0 0.0 0:00.02 init
87 anhanh 20 0 2888 1028 928 S 0.0 0.0 0:00.00 sh
88 anhanh 20 0 2888 1024 928 S 0.0 0.0 0:00.00 sh
113 anhanh 20 0 2888 1048 948 S 0.0 0.0 0:00.00 sh
117 anhanh 20 0 1338512 107772 44228 S 0.0 2.8 0:59.67 node
235 root 20 0 1836 108 0 S 0.0 0.0 0:00.00 init
236 root 20 0 1836 120 0 S 0.0 0.0 0:01.67 init
237 anhanh 20 0 1003768 59332 38008 S 0.0 1.5 0:04.02 node
244 root 20 0 1836 108 0 S 0.0 0.0 0:00.00 init
245 root 20 0 1836 120 0 S 0.0 0.0 0:02.96 init
246 anhanh 20 0 995792 53612 37996 S 0.0 1.4 0:07.09 node
256 anhanh 20 0 11.7g 597516 51888 S 0.0 15.3 3:19.31 node
268 anhanh 20 0 1240816 53836 40376 S 0.0 1.4 0:01.39 node
321 anhanh 20 0 111948 62960 17940 S 0.0 1.6 0:13.28 cpptools
344 anhanh 20 0 248136 173580 14104 S 0.0 4.4 0:03.29 TabNine
393 anhanh 20 0 994956 51344 38896 S 0.0 1.3 0:00.71 node
443 anhanh 20 0 48888 9092 5948 S 0.0 0.2 0:02.27 WD-TabNine
456 anhanh 20 0 339244 198676 22204 S 0.0 5.1 0:15.11 TabNine
563 anhanh 20 0 910116 680096 8636 S 0.0 17.4 0:20.27 TabNine-deep-lo
683 anhanh 20 0 4266644 20336 10980 S 0.0 0.5 0:00.86 cpptools-srv
726 anhanh 20 0 1352088 74716 40568 S 0.0 1.9 0:13.75 node
737 anhanh 20 0 5176 4360 3556 S 0.0 0.1 0:00.29 bash
816 anhanh 20 0 992540 51384 38800 S 0.0 1.3 0:00.55 node
822 anhanh 20 0 992188 49896 38576 S 0.0 1.3 0:00.54 node

it007sh>
```