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

Họ và tên: Nguyễn Trần Bảo Anh
Mã số sinh viên: 22520066
Lớp: IT007.O21.CNVN.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: 9

**Lưu ý: Xuất báo cáo theo định dạng PDF, đặt tên theo cú pháp:*
<Tên nhóm>_LAB6.pdf

Source code

Lab06 > C lab06.c > main(void)

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

#define MAX_LINE 80
int number_of_args;
int fdi,fdo;
char *args[MAX_LINE /2+1];
int SaveStdin,SaveStdout;

// Cấp phát bộ nhớ cho mảng
void Allocate_Memory(char *args_aray[])
{
    for (int i=0;i<MAX_LINE /2+1;i++)
    {
        args_aray[i] = (char *)malloc(sizeof(char) * MAX_LINE /2);
        // Cấp phát bộ nhớ cókích thước MAX_LINE/2 và con trỏ trỏ tới mảng này
    }
}

// nhập lệnh
void EnterCommand(char command[])
{
    fgets(command,MAX_LINE,stdin);
    short int len =strlen(command); // len = độ dài chuỗi
    command[len -1] = 0;// bỏ ký tự xuống dòng trong chuỗi
}

// tách lệnh thành các thành phần
void Tokernizer(char *tokens[],char *source,char *delim,int *num_of_words)
{
    char *p=strtok(source,delim);
    // Tìm kiếm ký tự đầu tiên trong source mà không có trong delim
    // con trỏ *p trỏ đến chuỗi con trong source được tách ra từ các ký tự trong delim
    int count =0;
```

```
    while (p!= NULL)
    {
        strcpy(tokens[count], p); // sao chép chuỗi p trỏ đến vào tokens[count];
        count++; // tăng count 1 đơn vị
        p=strtok(NULL,delim); // tiếp tục tìm kiếm các chuỗi con trong source
    }
    *num_of_words =count;// gán số lượng chuỗi con vừa tìm được vào nWord
    return;
}

// kiểm tra kết thúc lệnh
void control_sig(int sign)
{
    printf("\n");
    fflush(stdout);
}

// chuyển hướng đầu ra vào file
void redirect_output()
{
    SaveStdout =dup(STDOUT_FILENO);
    // tạo một bản sao củafile descriptor STDOUT_FILENO rồi gán cho SaveStdout

    fdo =open(args[number_of_args -1], O_CREAT |O_WRONLY |O_TRUNC);
    // mởfile args[number_of_args - 1] với các quyền tạo, chỉnh sửa

    dup2(fdo,STDOUT_FILENO);
    // Sao chép file fdo vàoSTDOUT_FILENO và bất kỳ đầu ra nào được ghi và sẽ được chuyển hướng vào file này.

    free(args[number_of_args -2]); // giải phóng bộ nhớ

    args[number_of_args -2] = NULL;// Đặt args[number_of_args -2] = NULL
}

// chuyển hướng đầu vào (input redirection) trong một chương trình chạy
void redirect_input()
{
    SaveStdin =dup(STDIN_FILENO);
    // tạo một bản sao của file descriptorSTDIN_FILENO và gán cho SaveStdin

0  0
```

```
81      fdi =open(args[number_of_args -1], O_RDONLY);
82      // mở file args[number_of_args -1] chỉ với quyền đọc và gán cho fdi
83
84      dup2(fdi,STDIN_FILENO);
85      // sao chép file descriptor fdi vào STDIN_FILENO
86
87      free(args[number_of_args -2]);args[number_of_args -2] = NULL;
88      // giải phóng bộ nhớ đã cấp phát và gán thành NULL
89  }
90
91  // khôi phục lại stdout sau khi đã thực hiện chuyển hướng đầu ra (output redirection) vào một file.
92  void RestoreOut()
93  {
94      close(fdo); // đóng file và giải phóng tài nguyên
95
96      dup2(SaveStdout,STDOUT_FILENO);
97      // sao chép file descriptor SaveStdout vào STDOUT_FILENO
98
99      close(SaveStdout); // đóng file saved_sdtout và giải phóng tài nguyên
100 }
101
102 // khôi phục lại stdin sau khi đã thực hiện chuyển hướng đầu vào (input redirection) từ một file
103 void RestoreIn()
104 {
105     close(fdi); // đóng file và giải phóng tài nguyên
106     dup2(SaveStdin,STDIN_FILENO); // sao chép file descriptor saved_sdtin vào STDIN_FILENO
107     close(SaveStdin); // đóng file và giải phóng tài nguyên
108 }
109
110 // thực thi hai lệnh được kết nối bằng một pipe trong một chương trình
111 void Execute_pile(char *parsed[],char *parsedpipe[])
112 {
113     pid_t PID1,PID2;
114     int stat;
115     int fd[2];
116     PID1 =fork(); // tạo tiến trình con p1
117
118     if (PID1 <0)
119     {
120         // in ra lỗi nếu không thể tạo
```

```
121         printf("\nLỗi không thể tạo tiến trình con");
122         return;
123     }
124 
125     if (PID1 == 0)
126     {
127         // Khi ở trong tiến trình con p1, tạo 1 tiến trình con khác: p2
128         if (pipe(fd) < 0)
129         {
130             printf("\nTạo ống thất bại");
131             return;
132         }
133         PID2 =fork();
134         if (PID2 <0)
135         {
136             printf("\nLỗi không thể tạo tiến trình con");
137             exit(0);
138         }
139 
140         // Khi ở trong tiến trình con p2
141         if (PID2 == 0)
142         {
143             // đóng đầu đọc của pipe
144             close(fd[0]);
145             // sao chép đầu ghi của pipe vào STDOUT_FILENO
146             dup2(fd[1], STDOUT_FILENO);
147             if (execvp(parsed[0], parsed) < 0)
148             {
149                 printf("\nKhông thể thực thi");exit(0);
150             }
151         }
152         else
153         {
154             // Khi p1 đang thực thi
155             wait(NULL); // đợi tiến trình con p2 kết thúc
156             close(fd[1]); // đóng đầu ghi của pipe
157             dup2(fd[0], STDIN_FILENO); // sao chép đầu đọc của pipe vào STDIN_FILENO
158 
159             if (execvp(parsedpipe[0], parsedpipe) < 0)
160             {
⚠ 0  0
```

```
                printf("\nKhông thể thực thi");
                exit(0);
            }
        }
    }
    else
    {
        wait(NULL); // đợi p1 kết thúc
    }
}

// tìm kiếm ký tự pipe
int Find_pile_char(char *cmd,char *cmdpiped[])
{
    int i;
    for (i=0;i<2;i++)
    {
        cmdpiped[i] = strsep(&cmd,"|");
        // Tìm kiếm ký tự "|", khi thấy sẽ loại bỏ lý tự nàyrồi gán vào mảng

        if (cmdpiped[i] == NULL)
        break;
    }

    if (cmdpiped[1] == NULL)
        return 0;// trả về 0 khi không tìm thấy
    else
    {
        return 1;// Trả về 1 nếu tìm thấy
    }
}

// tách một chuỗi lệnh (cmd) thành các đối số riêng biệt dựa trên ký tự khoảng trắng và
// lưu chúng vào mảng parsedArg
void parseCommandLine(char *cmd,char *parsedArg[])
{
    int i=0;
    while (cmd != NULL)
    {
        parsedArg[i] = strsep(&cmd," ");
```

```c
        // tìm kiếm ký tự khoảng trắng đầu tiên trong cmd
        // thay thế nó bằng ký tự null và trả về con trỏ đến đầu chuỗi
        if (parsedArg[i][0] != 0)// Nếu chuỗi không rỗng, tăng i lên 1 đơn vị
            i++;
    }
    parsedArg[i] = NULL;
}

// Kiểm tra có ký tự pipe
int ExecuteString(char *cmd,char *parsed[],char *parsedpipe[])
{
    char *cmdpiped[2]; // Mảng 2 con trỏ lưu 2 chuỗi lệnh được tách ra
    int piped =0;// khởi tạo Piped = 0
    piped =Find_pile_char(cmd,cmdpiped); // gán Piped bằng giá trị trả về của hàm kiểm tra có ký tự Pipe hay không
    // Nếu có sẽ gán 1, ngược lại sẽ gán 0
    if (piped)// Nếu có, phần thứ nhất sẽ gán vào parse, phần thứ 2 sẽ gán vào parsedpipe
    {
        parseCommandLine(cmdpiped[0], parsed);
        parseCommandLine(cmdpiped[1], parsedpipe);
    }
    return piped;
}


int main(void)
{
    int count_HF =0;
    int should_run =1; /* flag to determine when to exit program*/
    char command[MAX_LINE]; // mảng lưu command
    char history_command[MAX_LINE][MAX_LINE]; // mảng lưu lịch sử lệnh
    char *First_Command[MAX_LINE /2+1]; // command đầu tiên của pipe
    char *Second_command[MAX_LINE /2+1]; // command thứ 2 của pipe
    int iPileExe =0;// cờ kiểm tra có thực hiện pipe
    pid_t pid;
    int iRedirectOut =0,iRedirectIn =0;// cờ chuyển hướng đầu ra và đầu vào
    signal(SIGINT,control_sig);
    while (should_run)
    {
        do
        {
```

```
241             printf("it007sh>");
242             fflush(stdout);
243             EnterCommand(command);
244         }while (command[0] == 0);
245 
246         if (strcmp(command,"HF") == 0)
247         {
248             // Kiểm tra lệnh nhập vào có phải yêu cầu lịch sử câu lệnh hay không
249             if (history_command[count_HF -1] == 0)
250             {
251                 // Nếu không có lệnh nào trong mảng history_command đã lưu, in ra NULL
252                 printf("NULL\n");continue;
253             }
254             else
255             {
256                 // Ngược lại, in ra thực thi câu lệnh này
257                 for (int i=0;i<= count_HF -1;i++)
258                 {
259                     printf("%s\n",history_command[i]);
260                     strcpy(command,history_command[i]);
261                 }
262             }
263         }
264         else
265         {
266             // Lưu câu lệnh trong mảng history_command
267             strcpy(history_command[count_HF], command);
268             count_HF++;
269         }
270         iPileExe =ExecuteString(command,First_Command,Second_command);
271         // iPileExe sẽ trả về 0 khi là câu lệnh đơn giản như ls, pwd,... và
272         // sẽ trả về 1 khi là câulệnh với input của lệnh đầu tiên sẽ là
273         // đầu vào của lệnh tiếp theo (có ký tự pipe (|)).
274         if (iPileExe == 0)
275         {
276             // Nếu không có ký tự pipe trong lệnh
277             Allocate_Memory(args); // cấp phát bộ nhớ cho mảng
278             strcpy(command,history_command[count_HF -1]); // sao chép chuỗi history_command vào Command
279             Tokernizer(args,command," ", &number_of_args); // tách command thành các đối số riêng biệt và lưu chúng vào mảng args
280             free(args[number_of_args]);args[number_of_args] = NULL;// đặt vị trí cùng là NULL
⚠ 0  0
```

```
281             if (strcmp(args[0], "exit") == 0)
282             {
283                 // Kiểm tra lệnh có phải là exit hay không, nếu phải thì sẽ kết thúc
284                 break;
285             }
286             else
287             {
288                 if (strcmp(args[0], "~") == 0)
289                 {
290                     // kiểm tra vị trí đầu tiên của mảng có phải là ~
291                     char *homedir =getenv("HOME"); // trả về đường dẫn của thu mục HOME và in ra
292                     printf("Home : %s\n",homedir);
293                 }
294                 else // cd (change directory)
295                 if (strcmp(args[0], "cd") == 0)
296                 {
297                     // so sánh chuỗi có phải là cd hay không
298                     char dir[MAX_LINE];strcpy(dir,args[1]); // Khai báo mảng và sao chép chuỗi vào mảng này
299                     if (strcmp(dir,"~") == 0)// Nếu chuỗi vừa được sao chép là ~, copy mảng là đường dẫn đến thư mục HOME
300                     {
301                         strcpy(dir,getenv("HOME"));
302                     }
303                     chdir(dir); // thay đổi thư mục hiện tại thành thư mục chỉ định như trong mảng
304                     printf("Thư mục hiện tại : ");
305                     getcwd(dir,sizeof(dir)); // lấy đường dẫn đầy đủ của thư mục và in ra nó
306                     printf("%s\n",dir);
307                 }
308                 else
309                 {
310                     int iParrent =strcmp(args[number_of_args -1], "&"); // Kiểm tra vị trí cuối cùng trong chuỗi có phải là & hay không
311                     // nếu đúng thì gán iParrent = 0, ngược lại gán iParrent != 0
312                     if (iParrent == 0)
313                     {
314                         // Nếu iParrent = 0
315                         free(args[number_of_args -1]); // giải phóng vùng nhớ và gán lại bằng NULL
316                         args[number_of_args -1] = NULL;
317                     }
318                     // Nếu có nhiều hơn 1 đối số và đối số cuối là > thì sẽ chuyển hướng đầu ra
319                     // thông qua hàm redirect_output()
320                     if ((number_of_args >1) && strcmp(args[number_of_args -2], ">") == 0)
⚠ 0  0                                                                                    Ln 255, C
```

```
                {
                    redirect_output();
                    iRedirectOut =1;// đánh dấu đã thực hiện
                }
                else
                // Nếu có nhiều hơn 1 đối số và đối số cuối là < thì sẽ chuyển hướng đầuvào thông qua hàm redirect_input()
                if ((number_of_args >1) && strcmp(args[number_of_args -2], "<") ==0)
                {
                    redirect_input();
                    iRedirectIn =1;// đánh dấu đã thực hiện
                }
                pid =fork(); // Tạo tiến trình con pid
                // thông báo lỗi nếu tạo không thành công
                if (pid <0)
                {
                    fprintf(stderr,"Lỗi không thể tạo tiến trình công\n");
                    return -1;
                }

                // Khi đang ở trong tiến trình con
                if (pid == 0)
                {
                    // thực thi lệnh và nếu lệnh thực hiện thành công, ret = 1, ngược lại ret =-1 và in ra lỗi
                    int ret =execvp(args[0], args);
                    if (ret == -1)
                    {
                        printf("Lệnh không hợp lệ\n");
                    }
                    exit(ret);
                }
                else
                {
                    // khi đang ở trong tiến trình cha
                    if (iParrent)
                    {
                        while (wait(NULL) != pid){}
                        // Tiến trình cha đợi tiến trình con kết thúc
                        if (iRedirectOut)
                        {
                            // Nếu cờ chuyển hướng đầu ra bật lên
```

⚠ 0 0

```
                            RestoreOut();
                            iRedirectOut =0;// khôi phục cờ chuyển hướng đầu ra về ban đầu
                        }

                        if (iRedirectIn)
                        {
                            // Nếu cờ chuyển hướng đầu vào bật lên
                            RestoreIn();
                            iRedirectIn =0;// khôi phục cờ chuyển hướng đầu vào về ban đầu
                        }
                    }
                }
            }
        }
        else
        // Ngược lại, khi có ký tự pipe trong lệnh, thực hiện lệnh kết nối bằng pipe
        // thông qua hàm Execute_pile
        Execute_pile(First_Command,Second_command);
    }
    return 0;
}
```

6.4. BÀI TẬP THỰC HÀNH

1. Câu 1

Khi muốn thoát chương trình thì gõ lệnh "exit"

```
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab06$ ./lab06
it007sh>ls -l
total 88
-rwxr-xr-x 1 nguyentranbaoanh-22520066 nguyentranbaoanh-22520066 16728 May 28 11:01 bai1
-rw-r--r-- 1 nguyentranbaoanh-22520066 nguyentranbaoanh-22520066  3345 May 28 11:01 bai1.c
-rwxr-xr-x 1 nguyentranbaoanh-22520066 nguyentranbaoanh-22520066 21928 Jun 10 22:24 lab06
-rw-r--r-- 1 nguyentranbaoanh-22520066 nguyentranbaoanh-22520066 14795 Jun 10 22:24 lab06.c
-rwxr-xr-x 1 nguyentranbaoanh-22520066 nguyentranbaoanh-22520066 16392 May 28 11:24 test
-rw-r--r-- 1 nguyentranbaoanh-22520066 nguyentranbaoanh-22520066  1057 May 28 11:24 test.c
it007sh>echo 123
123
it007sh>exit
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab06$
⚠ 0  0
```

Khi lệnh thực thi không đúng sẽ báo lỗi

```
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab06$ ./lab06
it007sh>abc
Lệnh không hợp lệ
it007sh>cd..
Lệnh không hợp lệ
```

Giải thích kết quả:

- Với lệnh cat abc.txt, trong file abc.txt có dòng text: Nguyen Thi Ngoc Tram. Ban đầu, chương trình sẽ copy lệnh trong trong history_command (mảng lưu trữ lịch sử lệnh) và kiểm tra lệnh có ký tự pipe hay không. Tất nhiên đây là lệnh đơn giản, không có ký tự pipe nên giá trị iPipeExe ở đây gán là 0.
- Tiếp đến chương trình cấp phát bộ nhớ và tách lệnh thành các đối số riêng biệt thông qua hàm Tokernizer.
- Tiếp theo, bước đầu ta kiểm tra lệnh có chứa 'exit' hay không. Nếu có thì sẽ kết thúc chương trình. Tiếp đến sẽ kiểm tra lệnh có chứa ký tự '~' hay cd. Nếu có chương trình sẽ trả về đường dẫn Home hoặc đường dẫn đầy đủ của thư mục tương ứng với các lệnh.

- Nếu không phải các lệnh trên, chương trình sẽ tạo ra tiến trình con thực thi lệnh thông qua lệnh execvp(). Nếu thành công ret được gán sẽ có giá trị là 1 và ngược lại lệnh sẽ không thực thi được và ret trả về -1 – thông báo lỗi. Ở đây, tiến trình con sẽ thực hiện lệnh cat abc.txt và in ra dòng text của file: Nguyen Thi Ngoc Tram như trên màn hình.
- Cuối cùng, tiến trình cha sẽ đợi tiến trình con kết thúc và trả các giá trị cờ hiệu về giá trị ban đầu, giải phóng bộ nhớ và kết thúc, in ra it007> chờ người dùng nhập lệnh tiếp theo. Nếu lệnh tiếp theo là exit thì sẽ kết thúc luôn chương trình.
- Tương tự với các lệnh tiếp theo.

2. Câu 2

```
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab06$ ./lab06
it007sh>ls
bai1  bai1.c  lab06  lab06.c  test  test.c  test.txt
it007sh>echo abc
abc
it007sh>pwd
/home/nguyentranbaoanh-22520066/Lab06
it007sh>HF
ls
echo abc
pwd
/home/nguyentranbaoanh-22520066/Lab06
it007sh>
⚠ 0  0
```

Giải thích kết quả:

- Ban đầu, ta thực hiện câu lệnh cat abc.txt, chương trình sẽ lưu lệnh này trongmảng history_command và biến đếm số lệnh lúc này Count_HF sẽ là 1.
- Tiếp đến ta thực hiện lệnh thứ 2 echo abc, chương trình sẽ lưu lệnh này trongmảng history_command và biến đếm số lệnh lúc này Count_HF sẽ là 2.
- Kế đến ta thực hiện lệnh thứ 3 là ls, chương trình sẽ lưu lệnh này trong mảnghistory_command và biến đếm số lệnh lúc này Count_HF sẽ là 3.

- Cuối cùng ta nhập vào bàn phím HF, chương trình kiểm tra và thấy chuỗi trên đang yêu cầu lịch sử câu lệnh. Kết quả chương trình in ra các câu lệnh đã sử dụngvà thực hiện lại câu lệnh gần nhất là lệnh ls. Do đó ta có thể thấy kết quả in ra màn hình là 3 câu lệnh đã được sử dụng trước đó và danh sách các thư mục thông qua lệnh ls.

3. Câu 3

Test 1: ls > test1.txt

```
nguyentranbaoanh-22520066@LAPTOP-BPN8GDKT:~/Lab06$ ./lab06
it007sh>touch test1.txt
it007sh>ls > test1.txt
it007sh>cat test1.txt
bai1
bai1.c
lab06
lab06.c
test
test.c
test.txt
test1.txt
it007sh>
```

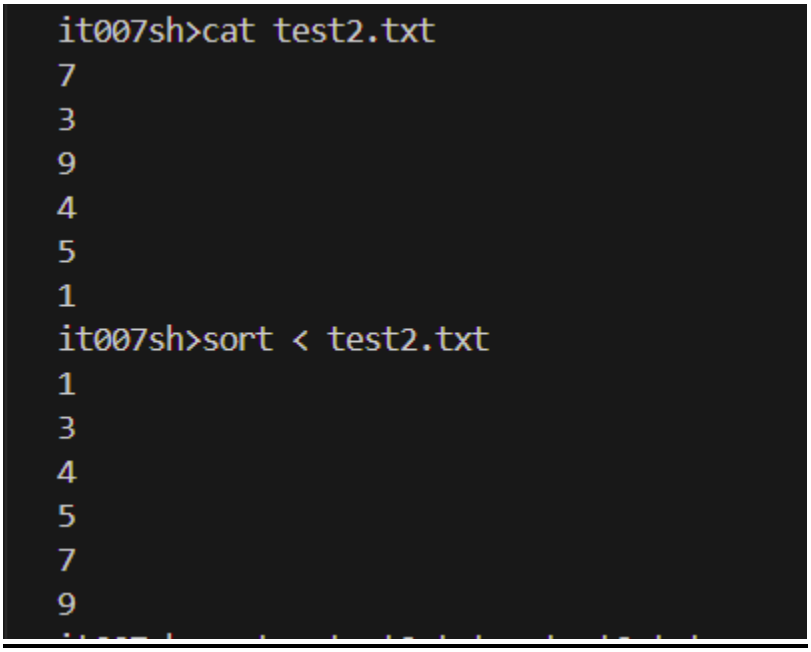
Giải thích kết quả:

- Với lệnh ls > test1.txt, ta thấy ban đầu file test1.txt là file trống. Khi ta thực hiện lệnh này, trước tiên chương trình sẽ kiểm tra có ký tự pipe ("|") hay không? Ở trường hợp này sẽ là không và sẽ gán cho iPipeExe là 0 và không thực hiện lệnh chứa pipe.
- Khi đó, ta sẽ cấp phát bộ nhớ cho mảng chứa tập lệnh, copy vào mảng chứa lịch sử tập lệnh và tách chúng thành các đối số riêng biệt thông qua hàm Tokernizer và lưu lại. Khi đó số lượng đối số của lệnh trên là 3 (ls, test1.txt, >) và chương trình sẽ kiểm tra trong đối số này có ký tự ">" nên sẽ thực hiện hàm redirect_output().
- Trong hàm redirect_output(), trước khi thực hiện lệnh ls in ra các thư mục, chương trình sẽ tạo 1 bản sao của file stdout là SaveStdout. Sau đó, chương trình sẽ mở file test1.txt với các quyền tạo, chỉnh sửa. Và sẽ sao chép tất cả các thư mục trong

lệnh ls thực thi vào test1.txt (sau khi đã thực hiện hàm này) và đánh dấu cờ hiệu iRedirectout đã thực hiện lên 1.

- Do đó, ta sẽ không thấy kết quả in ra màn hình mà các kết quả đã được chuyển hướng vào file test1.txt. Để kiểm tra, ta mở file test1.txt và thấy các thư mục đã được lưu vào đó.

Test 2: sort < test2.txt



Giải thích kết quả:

- Với lệnh sort < test2.txt, ta thấy ban đầu file test2.txt có các chữ số sắp xếp không theo thứ tự: 6,3,1,5,9,8,0. . Khi ta thực hiện lệnh này, trước tiên chương trình sẽ kiểm tra có ký tự pipe ("|") hay không? Ở trường hợp này sẽ là không và sẽ gán cho iPipeExe là 0 và không thực hiện lệnh chứa pipe.
- Khi đó, ta sẽ cấp phát bộ nhớ cho mảng chứa tập lệnh, copy vào mảng chứa lịch sử tập lệnh và tách chúng thành các đối số riêng biệt thông qua hàm Tokernizer và lưu lại. Khi đó số lượng đối số của lệnh trên là 3 (sort, test2.txt, <) và chương trình sẽ kiểm tra trong đối số này có ký tự "<" nên sẽ thực hiện hàm redirect_input().
- Trong hàm redirect_input(), trước khi thực hiện lệnh sort để sắp xếp các phần tử

trong thư mục test2.txt, chương trình sẽ tạo 1 bản sao của file stdin là SaveStdin. Sau đó, chương trình sẽ mở file test2.txt với quyền đọc. Sau đó sẽ thực hiện lệnh dup2() để sao chép file sao chép file descriptor vào stdin. Có nghĩa là bất cứ đầu đọc nào từ stdin đều sẽ chuyển hướng sang file này. Và đánh dấu cờ hiệu iRedirectin đã thực hiện lên 1.

- Điều này có nghĩa ta thực hiện hàm sort với đầu đọc được chuyển hướng đến file trên chứa các phần tử trong file test2.txt. Do đó kết quả trên màn hình ta có thể thấy các phần tử trong test2.txt đều đã được sắp xếp theo thứ tự từ bé đến lớn: 0,1,3,5,6,8,9.

4. Câu 4

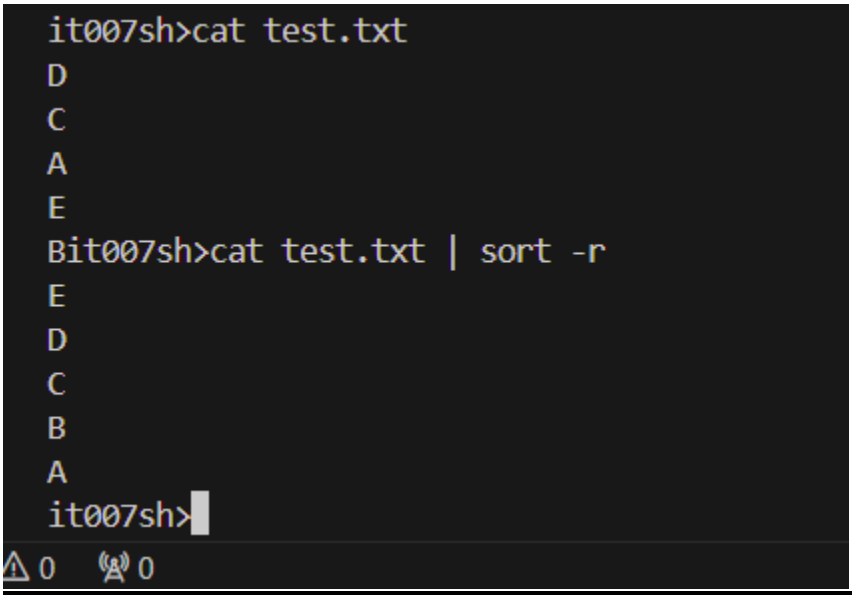
Test 1:

```
it007sh>cat test.txt
D
C
A
E
Bit007sh>cat test.txt | sort
A
B
C
D
E
it007sh>
```

Test 2:

```
it007sh>ls | grep test
test
test.c
test.txt
test1.txt
test2.txt
test3.txt
it007sh>ls
bai1  bai1.c  lab06  lab06.c  test  test.c  test.txt  test1.txt  test2.txt  test3.txt
it007sh>
```

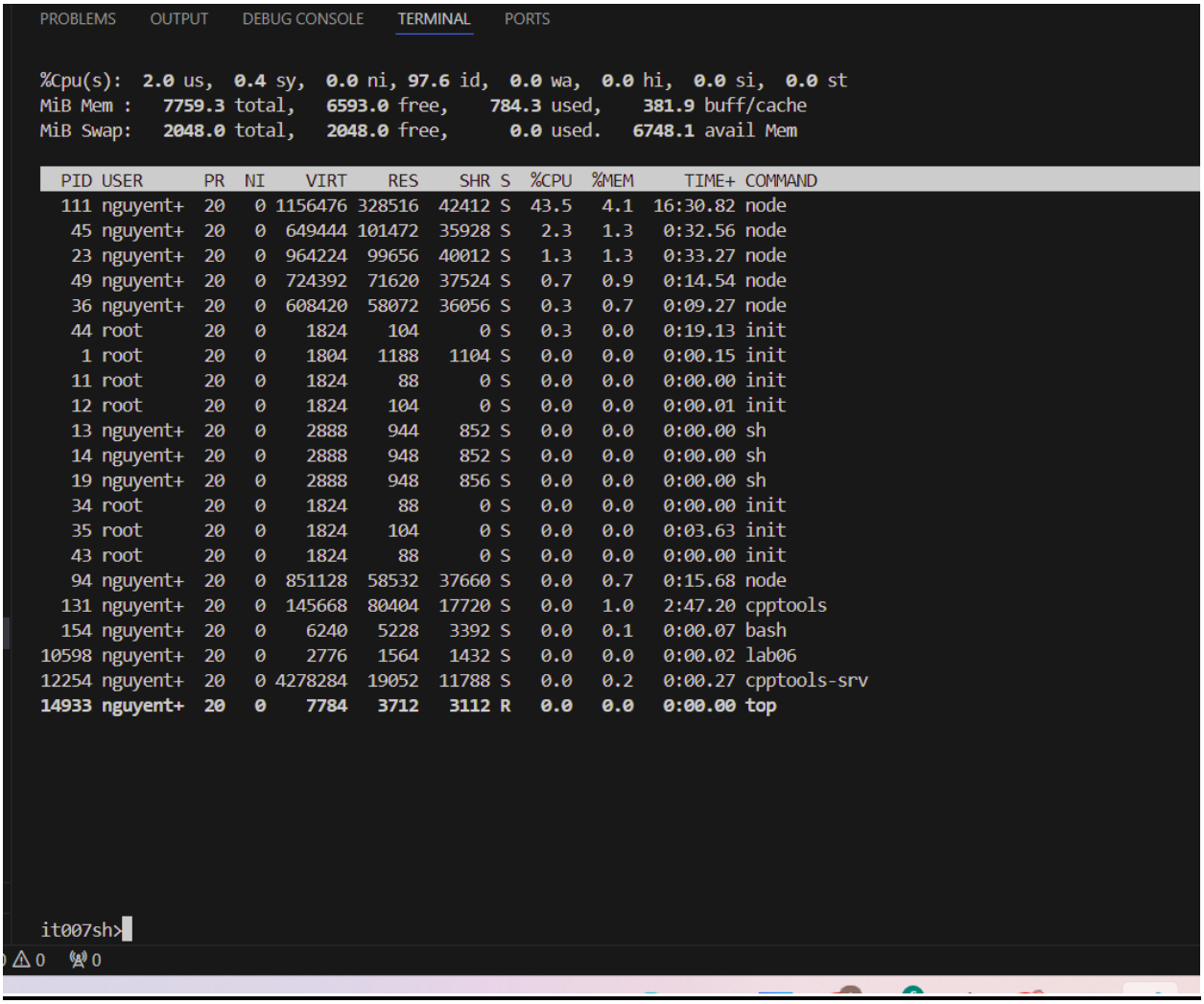
Test 3:



Giải thích kết quả:

- Như ở Testcase 1, với lệnh cat out.txt | sort, đầu tiên chương trình sẽ kiểm tra lệnh này có ký tự pipe (“|”) hay không thông qua hàm ExecuteString, parseCommandLine, Find_pile_char. Ở đây, hàm này sẽ trả giá trị về 1 rồi gán cho iPipeExe và sẽ lưu vào parse ở phần thứ nhất và parsepipe ở phần thứ 2.
- Tiếp đến, chương trình sẽ kiểm tra giá trị iPipeExe ở trên và thực hiện hàm Execute_pipe.
- Ở hàm này, đầu tiên sẽ tạo ra một tiến trình con p1. Khi tiến trình con p1 thực thi sẽ tạo ra 1 tiến trình con p2 và 1 pipe. Khi tiến trình p2 thực thi sẽ đóng đầu đọc của pipe và đầu ghi của pipe được sao chép vào stdout thông qua hàm dup2(). Khi đó lệnh cat out.txt sẽ được thực thi.
- Tiếp đến, khi tiến trình con p2 kết thúc, khi đó đầu ghi của pipe sẽ đóng lại và đầu đọc lại được mở và sao chép vào stdin thông qua hàm dup2(). Lúc này, lệnh sort sẽ thực hiện sắp xếp các phần tử có trong out.txt đã được đọc thông qua pipe rồi cuối cùng in ra màn hình kết quả các phần tử đã được sắp xếp tăng dần (ở đây theo chữ cái đầu, nếu trùng sẽ xét đến chữ cái kế tiếp).

5. Câu 5



Giải thích kết quả:

- Đầu tiên, khi nhập Ctrl+C, terminal sẽ gửi một tín hiệu ngắt đến tiến trình hiện tại đang chạy.

- Khi tiến trình nhận được tín hiệu ngắt, tín hiệu này sẽ được xử lý thông qua hàm control_sig với số hiệu signal(ở đây là 2 khi nhập Ctrl+C). Khi đó, hàm này sẽ dừng tiến trình hiện tại, xả bộ đệm và in ra một dòng it007> mới và tiếp tục chạy.