

운영체제론 개인 과제 1 보고서

제출자 : 이찬영 (2019098068, ICT 융합학부)

1. 본인이 작성한 함수에 대한 설명

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <unistd.h>
5  #include <string.h>
6  #include <strings.h>
7  #include <sys/wait.h>
8  #include <fcntl.h>
9
10 #define MAX_LINE 80      /* 명령어의 최대 길이 */
11
12 void redirection(char *temp[], int start, int argc) {
13
14     char *input_file = NULL;      // 입력 파일 이름을 저장할 포인터 변수
15     char *output_file = NULL;     // 출력 파일 이름을 저장할 포인터 변수
16     int input_fd = -1;            // 입력 파일 디스크립터
17     int output_fd = -1;           // 출력 파일 디스크립터
18
19     for (int i = start; i < argc; i++) {
20         if (strcmp(temp[i], "<") == 0) {
21             input_file = temp[i+1];
22             temp[i] = NULL;
23         }
24         else if (strcmp(temp[i], ">") == 0) {
25             output_file = temp[i+1];
26             temp[i] = NULL;
27         }
28         else if (temp[i] == NULL || strcmp(temp[i], "|") == 0) {
29             break;
30         }
31     }
32
33     if (input_file != NULL) {
34         input_fd = open(input_file, O_RDONLY);
35         if (input_fd == -1) {      // 파일을 오픈하는데 실패함
36             perror("open");
37             exit(EXIT_FAILURE);
38         }
39     }
40
41     if (output_file != NULL) {
42         output_fd = open(output_file, O_WRONLY);
43         if (output_fd == -1) {
44             perror("open");
45             exit(EXIT_FAILURE);
46         }
47     }
48
49     if (temp[start] != NULL) {
50         dup2(input_fd, 0);
51     }
52
53     if (temp[start+1] != NULL) {
54         dup2(output_fd, 1);
55     }
56
57     close(input_fd);
58     close(output_fd);
59 }
```

(CODE LINE : 12~)

Redirection 함수 - 리다이렉션을 실행합니다.

인자 : *temp[] (포인터 배열), start (탐색 시작 지점), argc (탐색 마지막 지점)

포인터 배열 *temp[] 위의 start 위치부터 argc 위치까지 문자열을 탐색하고 "<"나 ">" 문자가 있으면 리다이렉션을 수행합니다.

만약 중간에 "|" 문자나 NULL을 만나면 탐색을 중지하고 함수를 종료합니다.

(참고 자료 출처 : <https://woorld52.tistory.com/11>)

(CODE LINE : 19~)

temp[]배열에서 리다이렉션 연산자를 찾기 위한 반복문이 이어집니다.

start 인덱스부터 시작하여 리다이렉션 연산자("<", ">")가 나올 때마다

input_file 또는 output_file을 설정하고 temp[] 배열에서 해당 인자를 NULL로 설정합니다.

만약 중간에 NULL이나, "|" 을 만나면 반복문을 종료하게 됩니다.

(CODE LINE : 33~)

입력 파일이 설정되어 있으면 해당 파일을 열고 파일 디스크립터를 업데이트 합니다.

파일을 열 수 없으면 오류 메시지를 출력하고 프로그램을 종료합니다.

```
38     }
39     dup2(input_fd, STDIN_FILENO);
40     close(input_fd);
41 }
42
43 if (output_file != NULL) {
44     output_fd = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, 0644);
45     if (output_fd == -1) { // 파일을 출력하는데 실패함
46         perror("open");
47         exit(EXIT_FAILURE);
48     }
49     dup2(output_fd, STDOUT_FILENO);
50     close(output_fd);
51 }
52 }
53
54 static void cmdexec(char *cmd)
55 {
56     char *argv[MAX_LINE/2+1]; /* 명령어 인자를 저장하기 위한 배열 */
57     int argc = 0; /* 인자의 개수 */
58     char *p, *q; /* 명령어를 파싱하기 위한 변수 */
59
60     p = cmd; p += strspn(p, " \t");
61     do {
62
63         q = strpbrk(p, "\t'\"");
64
65         if (q == NULL || *q == ' ' || *q == '\t') {
66             q = strsep(&p, " \t");
67             if (*q) argv[argc++] = q;
68         }
69
70         else if (*q == '\\') {
71             q = strsep(&p, "\\");
72             if (*q) argv[argc++] = q;
73             q = strsep(&p, "\\");
74             if (*q) argv[argc++] = q;
```

(CODE LINE : 43~)

출력 파일이 설정되어 있으면 해당 파일을 열고 파일 디스크립터를 업데이트합니다.

파일을 열 수 없으면 오류 메시지를 출력하고 프로그램을 종료합니다.

(CODE LINE : 54~84)

기존에 있던 tsh.skeleton 코드입니다. (cmdexec() 함수)

```
75     }
76
77     else {
78         q = strsep(&p, "\\");
79         if (*q) argv[argc++] = q;
80         q = strsep(&p, "\\");
81         if (*q) argv[argc++] = q;
82     }
83 } while (p);
84 argv[argc] = NULL;
85
86
87 pid_t pid, pid2;          // 자식 프로세스 생성을 위한 프로세스 ID
88 int state = 0;            // 파이프("|")가 있는 상태와 파이프가 없는 상태를 구분해주기 위한 상태 변수
89 int start_redi = 0;       // 리다이렉션을 시작할 위치를 받는 변수
90 int a=0;
91
92 for(int k=0; k<argc; k++) {          // for문을 통해 argv 배열에 파이프("|") 문자가 있는지 확인한다. 만약 존재하면 state == 1, 존재하지 않으면 state == 0 이다.
93     if(strcmp(argv[k], "|") == 0) {
94         state = 1;
95         break;
96     }
97 }
98
99 if(state == 1) {          // "|" 파이프가 명령어에 존재하는 경우
100
101     for(int i=0 ; i<argc; i++) {
102         if(strcmp(argv[i], "|") == 0) {
103
104             argv[i] = NULL;          // argv를 탐색하는 중, 파이프("|")가 있으면, 해당 자리를 NULL로 만든다.
105
106             int fd[2];                // 파이프를 만들기 위한 배열 생성
107
108             if (pipe(fd) == -1) {      // 파이프 생성 중 오류 처리
109                 printf("pipe error\n");
110                 exit(EXIT_FAILURE);
111             }
112         }
113     }
114 }
```

(CODE LINE : 87~)

Argv 배열에 저장된 명령어들을 바탕으로 반복문을 수행하면서 파이프("|")가 있는 명령문과 파이프가 없는 명령문으로 state를 나눕니다. 다시 말해, 파이프가 있는 명령문은 state==1 조건문을 수행하게 되며, 파이프가 없는 명령문은 state==0인 조건문을 수행하게 됩니다.

(CODE LINE : 99~)

명령어에 파이프가 있는 경우, argv 배열에서 다시 for 문을 통해 탐색을 하고, 파이프("|")가 있는

위치를 찾아 NULL로 만들고, 파이프를 생성합니다.

(참고 자료 출처 : <https://daeuungcode.tistory.com/69>)

(참고 자료 출처 : <https://hump-mountain.tistory.com/4>)

```
111     }
112
113     pid = fork();                // 손자 프로세스를 생성해 손자 프로세스는 WRITE, 자식 프로세스는 READ를 수행함
114
115     if (pid == -1) {             // 손자 프로세스를 생성 중 오류 처리
116         perror("fork");
117         exit(EXIT_FAILURE);
118     }
119
120     if (pid == 0) {
121         close(fd[0]);
122         dup2(fd[1], STDOUT_FILENO);
123         close(fd[1]);
124         execvp(argv[start_redi], &argv[start_redi]);    //
125     }
126
127     else {
128         wait(NULL);
129         close(fd[1]);
130         dup2(fd[0], STDIN_FILENO);
131         close(fd[0]);
132
133         for(int j=i+1; j<argc; j++) {
134             if(strcmp(argv[j], "<") == 0 || strcmp(argv[j], ">") == 0) {
135                 redirection(argv, i+1, argc);
136                 break;
137             }
138
139             if(strcmp(argv[j], "|") == 0 || argv[j] == NULL) { break; }
140
141         }
142
143         pid2 = fork();
144         if(pid2 == 0) {
145             execvp(argv[i+1], &argv[i+1]);
146         }
147     }
```

(CODE LINE : 120~)

손자 프로세스입니다.

자식 프로세스와 파이프를 통해 데이터를 주고받는 프로세스 간 통신을 구현합니다.

해당 프로세스에서는 write를 수행하므로 사용하지 않는 fd[0]은 닫고, fd[1] 파이프를 표준 출력 파일 디스크립터로 복제하게 됩니다.

복제한 뒤에는 fd[1] 파이프의 출력 부분을 닫아주고, execvp()를 통해 argv[start_redi] 이후 부분을 실행합니다.

(CODE LINE : 127~)

자식 프로세스입니다.

wait(NULL)을 통해 손자 프로세스가 종료될 때까지 기다립니다.

*

본격적으로 손자 프로세스와 파이프를 통해 데이터를 주고받는 프로세스 간 통신을 구현합니다.

손자 프로세스에서 사용하지 않는 fd[1] 파이프는 닫아주고 fd[0] 파이프를 표준 입력 파일 디스크립터로 복제하게 됩니다.

복제한 뒤에는 fd[0] 파이프의 입력 부분을 닫아주고, for문을 통해 자식 프로세스에서 "<", ">" 이 있는지를 검사하고 만약 문자가 있으면 리다이렉션을 수행한 뒤 반복문을 빠져나간다.

"|" 파이프가 있거나 해당 argv가 NULL 일 때에도 반복문을 빠져나간다.

(CODE LINE : 143~)

자식 프로세스에서 다음 for문 수행을 하기 위해 프로세스를 하나 더 복제합니다.

execvp() 함수 이후의 코드는 실행되지 않으므로 프로세스를 하나 더 복제해서 실행시켜 줍니다.

자식 프로세스에서는 새로운 프로세스가 끝날 때까지 기다립니다.

```
146     }
147
148     else{
149         wait(NULL);
150
151         a = start_redi;        // 뒤에 있을 redirection 시작 지점을 a 변수로 설정해 준다.
152     }
153
154 }
155 start_redi = i+1;           // start_redi를 업데이트 해 준다.
156 }                           // if(strcmp(argv[i], "|") == 0) 문 종료
157
158 else if(strcmp(argv[i], "<") == 0 || strcmp(argv[i], ">") == 0) {           // 손자 프로세스에서 "<", ">" 등의 리다이렉션 문자가 있을 시에 수행을 위한 코드이다.
159     redirection(argv, a, argc);
160 }
161
162 } //for 문 종료
163
164 } // if(state == 1) 문 종료
165
166 else {                       // state == 0 인 경우, 즉 파이프("|")가 명령어에 존재하지 않는 경우
167     redirection(argv, 0, argc); // 리다이렉션 수행 ">", "<" 가 명령어에 없더라도 일단 함수를 수행한다. (만약 "<", ">"가 없으면 함수에서 자동으로 걸러짐)
168     if(argc > 0) {
169         execvp(argv[0], &argv[0]); // execvp() 함수를 통해 새프로그램을 생성해 argv[0]부터 마지막까지 명령어를 순차적으로 수행한다.
170     }
171 }
172 }
173
174 /*
175 * 기능이 간단한 유닉스 셸인 tsh (tiny shell)의 메인 함수이다.
176 * tsh은 프로세스 생성과 파이프를 통한 프로세스간 통신을 학습하기 위한 것으로
177 * 백그라운드 실행, 파이프 명령, 표준 입출력 리다이렉션 일부만 지원한다.
178 */
179 int main(void)
180 {
181     char cmd[MAX_LINE+1]; // 명령어를 저장하기 위한 버퍼 */
182     int len;              // 입력된 명령어의 길이 */
```

(CODE LINE : 166~)

State == 0 인 경우, 즉 명령어에 파이프("|")가 없는 경우 조건문을 수행합니다.

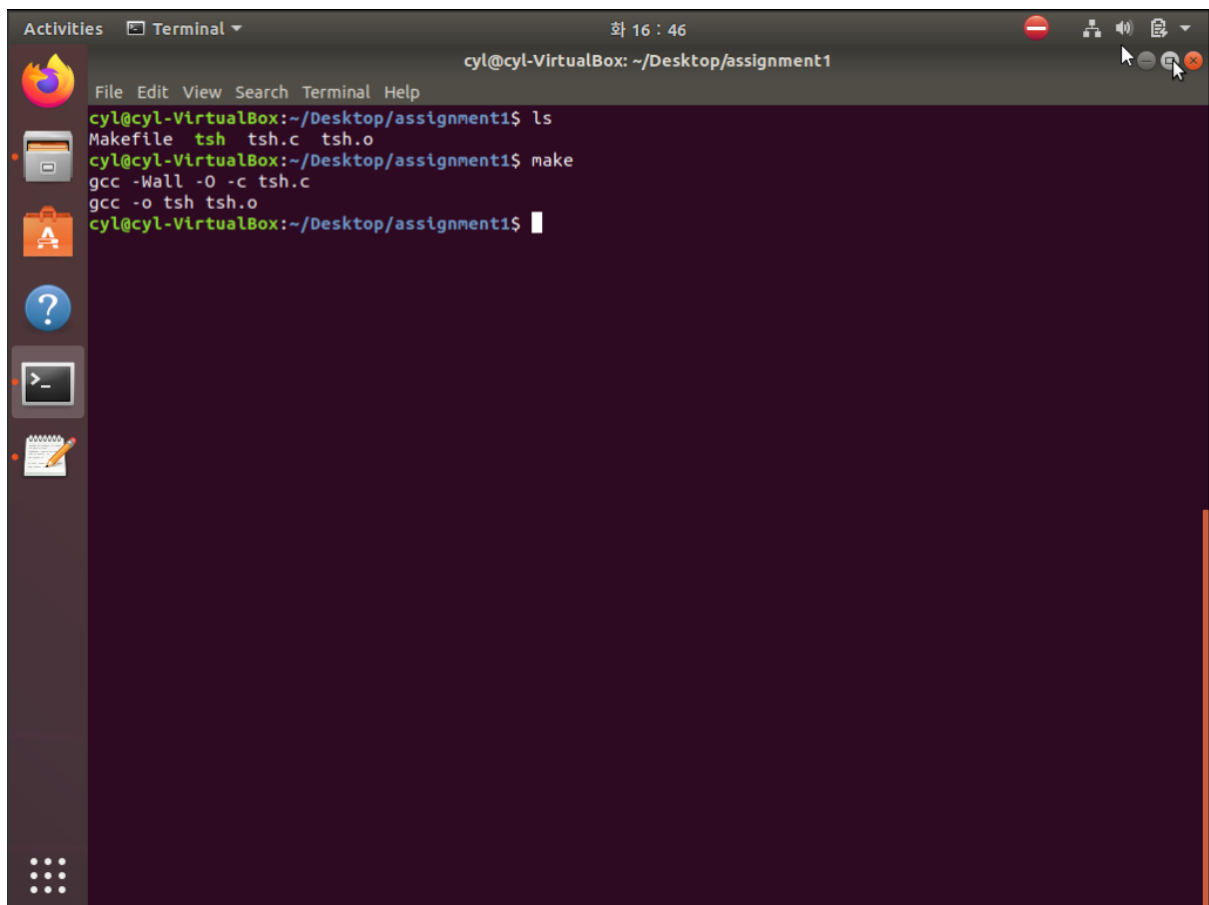
리다이렉션 명령어('<', '>')가 없더라도 일단 redirection() 함수를 수행한다. Redirection() 함수 내에서 리다이렉션 명령어('<', '>')가 없는 경우에 자동으로 걸리지게 됩니다.

리다이렉션 명령어('<', '>')가 있는 경우에는 리다이렉션을 수행합니다.

(CODE LINE : 174~)

이후는 tsh.skeleton.c 부분과 동일합니다.

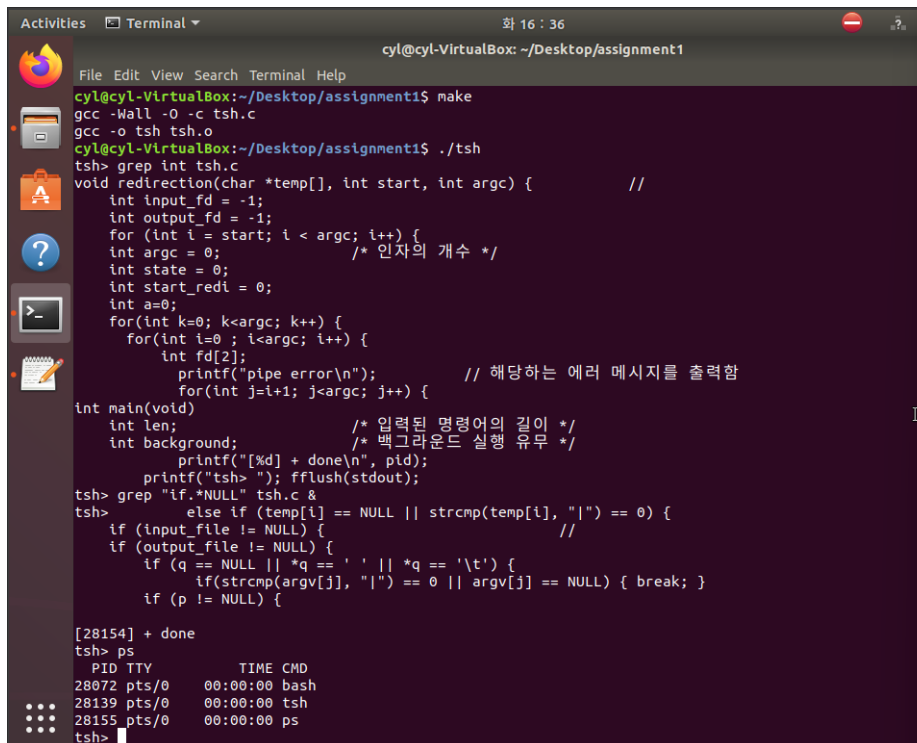
2. 컴파일 과정을 보여주는 화면 캡처

A screenshot of a Linux terminal window titled 'Terminal' with a subtitle 'cyl@cyl-VirtualBox: ~/Desktop/assignment1'. The terminal shows the execution of 'ls' and 'make' commands. The 'ls' command lists 'Makefile', 'tsh', 'tsh.c', and 'tsh.o'. The 'make' command runs 'gcc -Wall -O -c tsh.c' and 'gcc -o tsh tsh.o'. The terminal has a dark purple background and a light blue prompt. The window's title bar shows '화 16 : 46' and standard window controls. The left sidebar of the desktop environment is visible, showing icons for Firefox, Files, and other applications.

```
Activities Terminal
cyl@cyl-VirtualBox: ~/Desktop/assignment1
File Edit View Search Terminal Help
cyl@cyl-VirtualBox:~/Desktop/assignment1$ ls
Makefile  tsh  tsh.c  tsh.o
cyl@cyl-VirtualBox:~/Desktop/assignment1$ make
gcc -Wall -O -c tsh.c
gcc -o tsh tsh.o
cyl@cyl-VirtualBox:~/Desktop/assignment1$
```

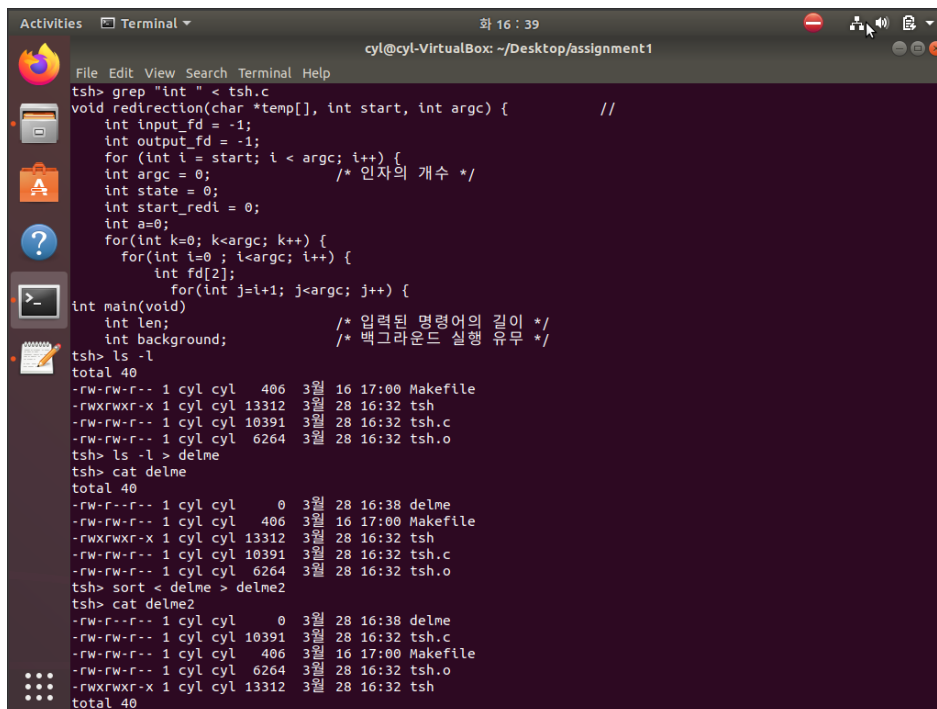
교수님께서 주신 Makefile로 make 명령어를 통해 tsh.c 파일을 컴파일한 화면을 캡처하였습니다. 아무 오류 없이 컴파일이 잘 수행된 것을 확인할 수 있습니다.

3. 실행 결과물에 대한 상세한 설명



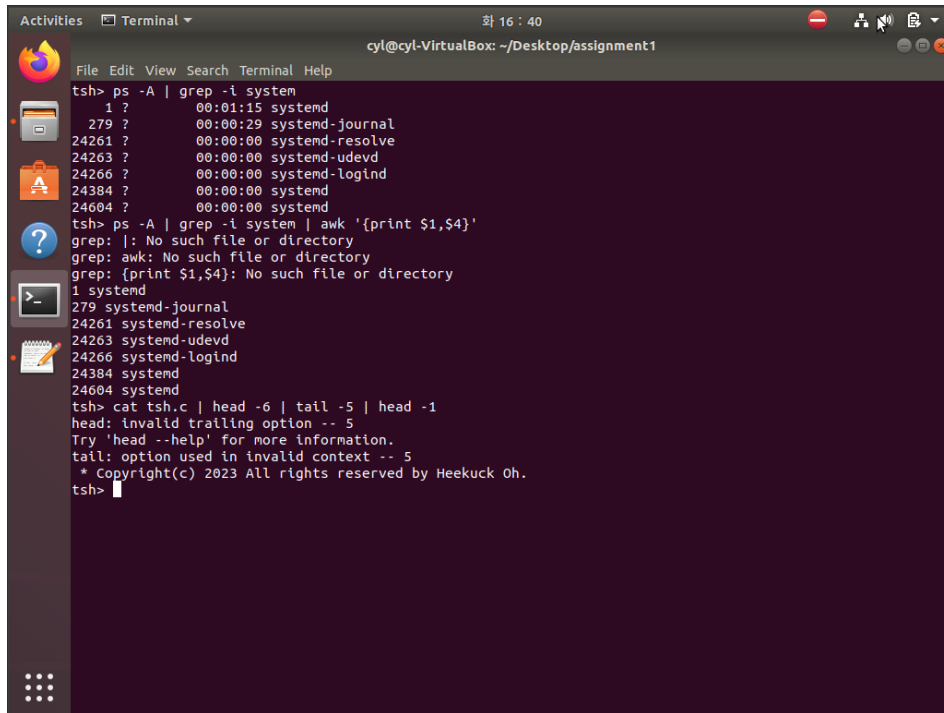
```
cyl@cyl-VirtualBox:~/Desktop/assignment1$ make
gcc -Wall -O -c tsh.c
gcc -o tsh tsh.o
cyl@cyl-VirtualBox:~/Desktop/assignment1$ ./tsh
tsh> grep int tsh.c
void redirection(char *temp[], int start, int argc) {
    int input_fd = -1;
    int output_fd = -1;
    for (int i = start; i < argc; i++) {
        int argc = 0;
        int state = 0;
        int start_redi = 0;
        int a=0;
        for(int k=0; k<argc; k++) {
            for(int i=0 ; i<argc; i++) {
                int fd[2];
                printf("pipe error\n");
                for(int j=i+1; j<argc; j++) {
int main(void)
int len;
int background;
printf("[%d] + done\n", pid);
printf("tsh> "); fflush(stdout);
tsh> grep "if.*NULL" tsh.c &
tsh>
else if (temp[i] == NULL || strcmp(temp[i], "|") == 0) {
if (input_file != NULL) {
if (output_file != NULL) {
if (q == NULL || *q == ' ' || *q == '\t') {
if (strcmp(argv[j], "|") == 0 || argv[j] == NULL) { break; }
if (p != NULL) {
[28154] + done
tsh> ps
PID TTY TIME CMD
28072 pts/0 00:00:00 bash
28139 pts/0 00:00:00 tsh
28155 pts/0 00:00:00 ps
tsh>
```

1. 기본 명령을 잘 수행합니다.



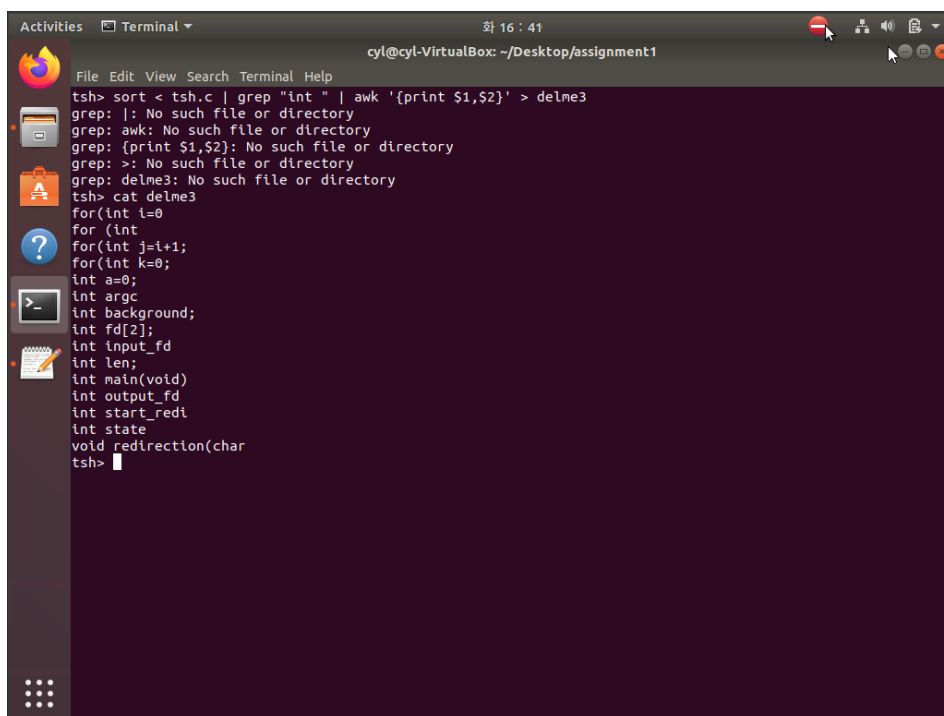
```
tsh> grep "int " < tsh.c
void redirection(char *temp[], int start, int argc) {
    int input_fd = -1;
    int output_fd = -1;
    for (int i = start; i < argc; i++) {
        int argc = 0;
        int state = 0;
        int start_redi = 0;
        int a=0;
        for(int k=0; k<argc; k++) {
            for(int i=0 ; i<argc; i++) {
                int fd[2];
                for(int j=i+1; j<argc; j++) {
int main(void)
int len;
int background;
tsh> ls -l
total 40
-rw-rw-r-- 1 cyl cyl 406 3월 16 17:00 Makefile
-rwxrwxr-x 1 cyl cyl 13312 3월 28 16:32 tsh
-rw-rw-r-- 1 cyl cyl 10391 3월 28 16:32 tsh.c
-rw-rw-r-- 1 cyl cyl 6264 3월 28 16:32 tsh.o
tsh> ls -l > delme
tsh> cat delme
total 40
-rw-r--r-- 1 cyl cyl 0 3월 28 16:38 delme
-rw-rw-r-- 1 cyl cyl 406 3월 16 17:00 Makefile
-rwxrwxr-x 1 cyl cyl 13312 3월 28 16:32 tsh
-rw-rw-r-- 1 cyl cyl 10391 3월 28 16:32 tsh.c
-rw-rw-r-- 1 cyl cyl 6264 3월 28 16:32 tsh.o
tsh> sort < delme > delme2
tsh> cat delme2
-rw-r--r-- 1 cyl cyl 0 3월 28 16:38 delme
-rw-rw-r-- 1 cyl cyl 10391 3월 28 16:32 tsh.c
-rw-rw-r-- 1 cyl cyl 406 3월 16 17:00 Makefile
-rw-rw-r-- 1 cyl cyl 6264 3월 28 16:32 tsh.o
-rwxrwxr-x 1 cyl cyl 13312 3월 28 16:32 tsh
total 40
```

2. 표준 입출력 리다이렉션을 잘 수행합니다.



```
File Edit View Search Terminal Help
tsh> ps -A | grep -i system
 1 ?      00:01:15 systemd
279 ?      00:00:29 systemd-journal
24261 ?    00:00:00 systemd-resolve
24263 ?    00:00:00 systemd-udev
24266 ?    00:00:00 systemd-logind
24384 ?    00:00:00 systemd
24604 ?    00:00:00 systemd
tsh> ps -A | grep -i system | awk '{print $1,$4}'
grep: |: No such file or directory
grep: awk: No such file or directory
grep: {print $1,$4}: No such file or directory
 1 systemd
279 systemd-journal
24261 systemd-resolve
24263 systemd-udev
24266 systemd-logind
24384 systemd
24604 systemd
tsh> cat tsh.c | head -6 | tail -5 | head -1
head: invalid trailing option -- 5
Try 'head --help' for more information.
tail: option used in invalid context -- 5
* Copyright(c) 2023 All rights reserved by Heekuck Oh.
tsh>
```

3. 파이프 기능을 잘 수행하지만, 다중 파이프 시 뒤에 있는 명령어를 파일이나 디렉토리 이름으로 본다는 문제가 있습니다.



```
File Edit View Search Terminal Help
tsh> sort < tsh.c | grep "int " | awk '{print $1,$2}' > delme3
grep: |: No such file or directory
grep: awk: No such file or directory
grep: {print $1,$2}: No such file or directory
grep: >: No such file or directory
grep: delme3: No such file or directory
tsh> cat delme3
for(int i=0
for (int
for (int
for(int j=i+1;
for(int k=0;
int a=0;
int argc
int background;
int fd[2];
int input_fd
int len;
int main(void)
int output_fd
int start_redi
int state
void redirection(char
tsh>
```

4. 조합 기능 역시 잘 수행하지만, 뒤에 있는 명령어를 파일이나 디렉토리 이름으로 본다는 문제가 있습니다.

4. 과제를 수행하면서 경험한 문제점과 느낀 점

'맨 처음에 접근을 반복문으로 한 것이 과제를 수행하는데 어려움을 더하지 않았나'라고 생각합니다.

반복문으로 주어진 기능을 구현하는 중에 굉장히 많은 고비들이 있었습니다. 저 같은 경우에는 명령어를 담은 포인터 배열인 argv 상에서 바로 명령어를 수행하게 하려고 하다 보니 더 그랬던 거 같습니다. 우선, 표준 입출력 리다이렉션까지는 비교적 수월하게 반복문으로 구현하였으나, 파이프의 기능을 구현하려고 하는 중에 반복문의 구조를 완전히 바꿔야 한다는 것을 깨달았고, 이를 오랜 시간 고민 끝에 해결하였습니다.

파이프는 프로세스를 복제하고, `execvp()` 함수를 통해 그것을 실행하기 때문에 argv의 다음 원소를 가리키게 하려면(반복문을 돌게 하려면) 제 수준에서는 프로세스를 하나 더 복제해서 `execvp()` 함수를 실행하는 방법 밖에 없다는 것을 깨닫게 되었습니다.

또한 파이프('|')의 역할과 의미를 정확히 이해하지 못한 것도 시행착오 중의 하나였습니다. 파이프는 단순 명령을 수행하는 것이 아닌, 프로세스를 복제하고 파이프를 통해 연결한다는 개념으로 생각했어야 함에도 불구하고, 단순히 리다이렉션('<', '>') 기호와 같은 수준으로 의미를 생각했습니다. 따라서 이것은 제가 다중 파이프를 구현하지 못한 가장 큰 장애물이었습니다.

여전히 아쉬움이 남는 것은 반복문에서 다중 파이프 명령문과 조합 명령문을 수행할 때 쓰는 `grep`: 오류를 완전히 해결할 수는 없는가 하는 것입니다. 이것을 단순히 반복문의 한계로 봐야 할지, 아니면 구현하지 못한 제 부족함으로 봐야 할지는 의문으로 남아 있습니다.

이번 과제 수행을 통해, 친구와 의견을 교환하고 나누는 것이 얼마나 큰 것인가를 다시 한 번 깨닫게 되었고 (제가 생각하지 못했던 지점들을 생각하게 만들어 주었고), 무작정 코딩을 바로 하기 보다는 큰 계획과 틀을 세우고, 어느 정도 pseudocode를 스스로 만들어 보고 나서 코딩을 하는 것이 더 효율적이라는 생각이 들었습니다. 뿐만 아니라 수업 시간 때 단순히 이론과 개념으로만 배웠던 프로세스와 파이프 개념(IPC)을 프로젝트 과제를 통해 시행착오를 거듭하면서 체득할 수 있었던 것이 가장 큰 소득이었습니다. 앞으로 남은 수업도 성실하고 열심히 임하겠습니다. 감사합니다.