

# 시 스템 소 프 트 웨 어 실 습

## - Project -

사용자로부터 입력받은 명령어 실행하기 (총 3점)	사용자로부터 명령어 입력받기 (1점)	○
	입력받은 명령어 실행하기 (1.5점)	○
	'q' 명령어를 입력받으면 Shell 종료하기 (0.5점)	○
ls 함수 구현하기 (총 1점)	현재 경로의 파일 및 디렉토리 목록 출력 (1점)	○
탭(Tab)키를 이용한 자동완성 기능 구현하기 (총 2점)	파일명 일부 입력 후 탭키를 한 번 눌렀을 경우 파일의 이름 자동완성하기 (2점)	X
탭(Tab)키를 이용한 파일 목록 출력 기능 구현하기 (총 2점)	지정한 경로에서 탭키를 두 번 눌렀을 경우 파일의 목록 출력하기 (2점)	○
표준 입력(<) 리다이렉션 구현하기 (총 1.5점)	'<' 기호와 명령어 분리하기 (0.5점)	○
	'<' 기호 뒤에 입력받은 파일로 표준 입력 대체 하기 (1점)	○
표준 출력(>) 리다이렉션 구현하기 (총 1.5점)	'>' 기호와 명령어 분리하기 (0.5점)	○
	'>' 기호 뒤에 입력받은 파일로 표준 출력 대체 하기 (1점)	○
파이프를 이용하여 둘 이상의 명령어 함께 실행하기 (총 4점)	' ' 기호 앞의 명령 결과를 ' ' 기호 뒤의 명령의 입력으로 전달하기 (1점)	○
	' ' 기호와 리다이렉션(<, >) 기호 함께 사용하기 (1점)	○
	파이프 사용 개수를 임의로 할 수 있도록 구현하기 (2점)	○
'&' 기호를 이용하여 백그라운드로 명령 실행하기 (총 3점)	'&' 기호와 명령어 분리하기 (0.5점)	○
	입력받은 명령어 백그라운드로 실행하기 (1점)	○
	파이프를 이용한 명령을 백그라운드로 실행하기 (1.5점)	○
'SIGINT', 'SIGQUIT' 시그널 무시하도록 구현하기 (총 2점)	시그널 함수를 'SIGINT', 'SIGQUIT' 동작 재정의하기 (2점)	○

학 번 : 2014707040

학 과 : 컴퓨터소프트웨어학과

이 름 : 유진혁

## □ Shell 기본 기능 및 ls 함수 구현하기

### ■ 사용자로부터 입력받은 명령어 실행하기 (총 3점)

#### □ 사용자로부터 명령어 입력받기 (1점)

```
do
{
    ch = getch();
    len = strlen(inbuf);
    if (ch == '\t')
    {
        if ((tabstr = strchr(inbuf, ' ')) == NULL)
            continue;
        tabstr = tabstr + 1;
        if ((ch = getch()) == '\t')
        {
            if((getcwd(pwd, 255) == NULL) || (chdir(tabstr) == -1))
                continue;
            printf("\n");
            myls();
            chdir(pwd);
            fputs("User Shell >> ", stdout);
            fputs(inbuf, stdout);
        }
    }
    else if (ch == 127)
    {
        if (len > 0)
        {
            inbuf[len - 1] = 0;
            printf("\b");
            printf(" ");
            printf("\b");
        }
    }
    else if ((ch == '\n') || ((ch >= 32) && (ch <= 126)))
    {
        inbuf[len] = ch;
        printf("%c", ch);
    }
} while (ch != '\n');
inbuf[len] = 0;
```

getch으로 사용자로부터 입력을 받고 문자를 입력 받았을 경우 명령어 문자열에 덧붙인다.

#### □ 입력받은 명령어 실행하기 (1.5점)

##### ■ 입력받은 명령어 실행 후 정상적으로 Shell로 복귀

```
else if (makeargv(s, " ", &chargv) <= 0)
    perror("Failed to parse command line");
else
{
    if ((chargv[1] == NULL) && (strcmp(chargv[0], "ls") == 0))
    {
        myls();
        exit(1);
    }
    else
    {
        execvp(chargv[0], chargv);
        perror("Failed to execute command");
    }
}
```

입력받은 명령어 문자열을 makeargv 함수(strtok을 이용)로 공백 기준으로 나누고 이를 문자열 배열로 만든다. execvp에 이 배열을 인자로 넣어 명령어를 실행시킨다.

#### □ 'q' 명령어를 입력받으면 Shell 종료하기 (0.5점)

```
if (strcmp(inbuf, "q") == 0)
    break;
```

입력받은 문자열과 "q" 를 비교하여 같을 경우 무한 루프를 탈출 시켜 Shell을 종료한다.

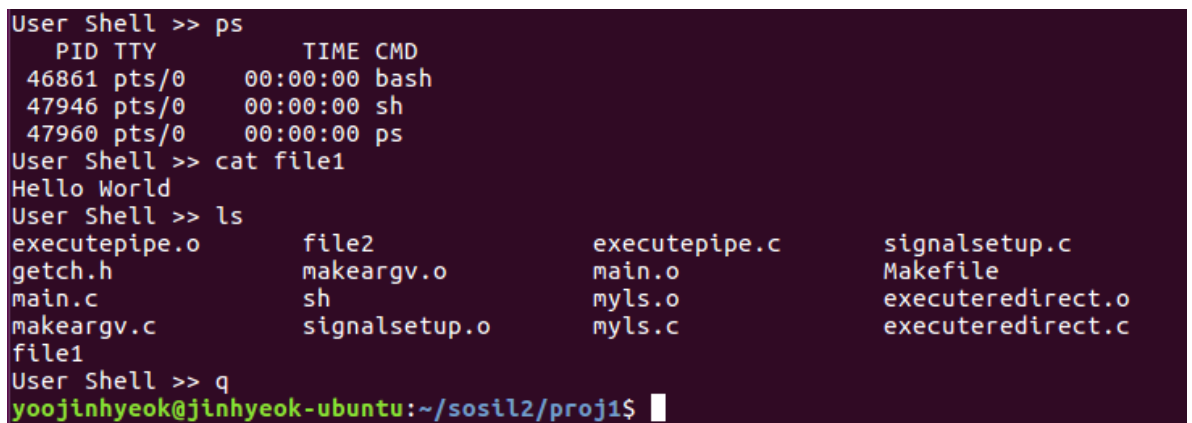
## ■ ls 함수 구현하기 (총 1점)

### □ 현재 경로의 파일 및 디렉토리 목록 출력 (1점)

```
int myls()
{
    char pwd[255];
    DIR* dirp;
    struct dirent* direntp;
    int count = 0;

    if (getcwd(pwd, 255) == NULL)
    {
        perror("Failed to get pwd.");
        return 1;
    }
    if ((dirp = opendir(pwd)) == NULL)
    {
        perror("Failed to open current directory.");
        return 1;
    }
    while ((direntp = readdir(dirp)) != NULL)
    {
        if (direntp->d_name[0] == '.')
            continue;
        else
        {
            printf("%-20s", direntp->d_name);
            count++;
        }
        if ((count % 4) == 0)
            printf("\n");
    }
    printf("\n");
    closedir(dirp);
    return 0;
}
```

현재 디렉토리를 열고 디렉토리 스트림 항목 중 파일 이름을 읽어 들여 출력한다. 옵션 인자 없는 ls 명령어가 입력되면 해당 함수가 호출된다.



```
User Shell >> ps
  PID TTY          TIME CMD
 46861 pts/0        00:00:00 bash
 47946 pts/0        00:00:00 sh
 47960 pts/0        00:00:00 ps
User Shell >> cat file1
Hello World
User Shell >> ls
executepipe.o      file2              executepipe.c      signalsetup.c
getch.h            makeargv.o         main.o             Makefile
main.c             sh                 myls.o             executeredirect.o
makeargv.c         signalsetup.o      myls.c             executeredirect.c
file1
User Shell >> q
yoojinhyeok@jinhyeok-ubuntu:~/sosil2/proj1$
```

명령어 실행 및 ls 함수 호출, Shell 종료 화면

### □ 자동완성(Tab 키) 기능 구현하기

## ■ 탭(Tab)키를 이용한 자동완성 기능 구현하기 (총 2점)

### □ 파일명 일부 입력 후 탭키를 한 번 눌렀을 경우 파일의 이름 자동완성하기 (2점)

■ 비슷한 이름의 파일이 여러 개일 경우 공통된 부분까지만 출력

■ Tip : 명령 프롬프트 내용 및 탭 키 입력은 getch() 함수를 이용

(구현하지 못함)

## ■ 탭(Tab)키를 이용한 파일 목록 출력 기능 구현하기 (총 2점)

### □ 지정한 경로에서 탭키를 두 번 눌렀을 경우 파일의 목록 출력하기 (2점)

```
if (ch == '\t')
{
    if ((tabstr = strchr(inbuf, ' ')) == NULL)
        continue;
    tabstr = tabstr + 1;
    if ((ch = getch()) == '\t')
    {
        if((getcwd(pwd, 255) == NULL) || (chdir(tabstr) == -1))
            continue;
        printf("\n");
        myls();
        chdir(pwd);
        fputs("User Shell >> ", stdout);
        fputs(inbuf, stdout);
    }
}
```

탭키를 누르면 현재 명령어 라인의 공백 문자 이후 문자열을 경로로 설정하고 한 번 더 누르면 그 경로로 이동하여 ls 호출 후 다시 원래 경로로 복귀한다.

```
User Shell >> ls
executepipe.o      file2      executepipe.c    signalsetup.c
getch.h            makeargv.o  main.o           Makefile
main.c             sh          myls.o           executeredirect.o
makeargv.c         signalsetup.o myls.c           executeredirect.c
file1
User Shell >> ls ./
executepipe.o      file2      executepipe.c    signalsetup.c
getch.h            makeargv.o  main.o           Makefile
main.c             sh          myls.o           executeredirect.o
makeargv.c         signalsetup.o myls.c           executeredirect.c
file1
User Shell >> ls ./
```

탭 2번을 이용한 목록 출력

### □ 리다이렉션 기능 구현하기

## ■ 표준 입력(<) 리다이렉션 구현하기 (총 1.5점)

### □ '<' 기호와 명령어 분리하기 (0.5점)

```
if ((infile = strchr(cmd, '<')) == NULL)
    return 0;
*infile = 0;
infile = strtok(infile + 1, " ");
if (infile == NULL)
    return 0;
```

'<' 기호를 찾아서 그 기호를 널 문자로 대체시킨다. 그 후 strtok 함수를 이용하여 앞뒤의 공백을 제거한다. 결국 남아있는 문자열은 리다이렉션으로 사용될 파일명이다.

### □ '<' 기호 뒤에 입력받은 파일로 표준 입력 대체 하기 (1점)

```
if ((infd = open(infile, O_RDONLY)) == -1)
    return -1;
if (dup2(infd, STDIN_FILENO) == -1)
{
    close(infd);
    return -1;
}
return close(infd);
```

파일을 열고 표준 입력을 해당 파일 디스크립터로 복사하여 리다이렉트 해준다.

## ■ 표준 출력(>) 리다이렉션 구현하기 (총 1.5점)

### □ '>' 기호와 명령어 분리하기 (0.5점)

```
if ((outfile = strchr(cmd, '>')) == NULL)
    return 0;
*outfile = 0;
outfile = strtok(outfile + 1, " ");
if (outfile == NULL)
    return 0;
```

표준 입력 리다이렉션과 마찬가지로 '>' 기호를 찾고 널로 대체시킨 뒤, 앞뒤 공백을 제거한다.

### □ '>' 기호 뒤에 입력받은 파일로 표준 출력 대체 하기 (1점)

```
if ((outfd = open(outfile, O_WRONLY | O_CREAT | O_TRUNC, 0664)) == -1)
    return -1;
if (dup2(outfd, STDOUT_FILENO) == -1)
{
    close(outfd);
    return -1;
}
return close(outfd);
```

표준 출력을 대체할 파일이 존재하지 않을 경우 파일을 생성한 뒤에 파일을 연다. 표준 출력을 해당 파일의 디스크립터로 복사하여 리다이렉트 해준다.

```
User Shell >> cat file1
D
B
C
A
User Shell >> sort < file1
A
B
C
D
User Shell >> cat file2
Hello
User Shell >> ls -l > file2
User Shell >> cat file2
합계 84
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 396 12월 19 21:43 Makefile
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 1232 12월 19 14:24 executepipe.c
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 3144 12월 19 21:48 executepipe.o
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 1741 12월 20 01:59 executeredirect.c
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 3784 12월 20 02:00 executeredirect.o
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 8 12월 20 02:06 file1
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 0 12월 20 02:07 file2
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 301 12월 15 17:46 getch.h
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 2629 12월 20 01:09 main.c
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 5552 12월 20 01:09 main.o
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 688 12월 19 21:42 makeargv.c
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 2136 12월 19 21:48 makeargv.o
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 570 12월 18 21:54 myls.c
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 2328 12월 19 21:48 myls.o
-rwxrwxr-x 1 yoojinhyeok yoojinhyeok 19504 12월 20 02:00 sh
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 711 12월 19 21:40 signalsetup.c
-rw-rw-r-- 1 yoojinhyeok yoojinhyeok 2376 12월 19 21:48 signalsetup.o
User Shell >>
```

표준 입력과 표준 출력 리다이렉션

### □ 파이프라인(|) 기능 구현하기

## ■ 파이프를 이용하여 둘 이상의 명령어 함께 실행하기 (총 4점)

□ '|' 기호 앞의 명령 결과를 '|' 기호 뒤의 명령의 입력으로 전달하기 (1점)

□ '|' 기호와 리다이렉션(<, >) 기호 함께 사용하기 (1점)

□ 파이프 사용 개수를 임의로 할 수 있도록 구현하기 (2점)

■ Tip : 재귀 함수 사용

```
count = makeargv(cmds, "|", &pipelist);
if (count <= 0)
{
    fprintf(stderr, "Failed to find any commands\n");
    exit(0);
}
for (i = 0; i < count - 1; i++)
{
    if (pipe(fds) == -1)
        perror_exit("Failed to find any commands\n");
    else if ((child = fork()) == -1)
        perror_exit("Failed to create process to run command");
    else if (child == 0)
    {
        if (dup2(fds[1], STDOUT_FILENO) == -1)
            perror_exit("Failed to connect pipeline");
        if (close(fds[0]) || close(fds[1]))
            perror_exit("Failed to close needed files");
        executeredirect(pipelist[i], i == 0, 0);
        exit(1);
    }
    wait(NULL);
    if (dup2(fds[0], STDIN_FILENO) == -1)
        perror_exit("Failed to connect last component");
    if (close(fds[0]) || close(fds[1]))
        perror_exit("Failed to do final close");
}
executerdirect(pipelist[i], i == 0, 1);
exit(1);
```

재귀 함수가 아닌 반복문을 이용하여 두 개 이상 명령이 연결되도록 하였다. '|' 기호를 구분자로 하여 명령어를 나눈 배열을 만들기 위해 makeargv 함수를 호출한다. 반복문 안에서 파이프와 자식 프로세스를 생성한다. 그 후 마지막 명령을 제외한 각 명령의 표준 출력을 다음 명령의 표준 입력으로 연결시킨다. 자식 프로세스는 표준 출력을 파이프로 복사하고, 리다이렉션을 거친 명령 실행 부분이 있는 executeredirect 함수를 호출한다. 부모 프로세스는 자식 프로세스의 진행을 기다렸다가 자신의 표준입력을 파이프로 복사하고 반복문의 처음으로 돌아간다. 파이프 라인의 마지막 명령은 반복문을 나온 부모 프로세스가 바로 실행한다.

과제 예시에는 ps | sort 명령이 호출되어 ps의 결과가 정렬되어 출력되었으나, 본인이 구현한 셸과 실제 bash 셸에선 해당 명령을 호출하여도 ps의 결과가 정렬되어 출력되지 않는다. 그러므로 다른 명령을 통해 파이프라인 기능의 구현을 보이겠다.

```
User Shell >> ps -l
F S  UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    46861  46856  0  80   0 -  7706 wait   pts/0      00:00:00 bash
0 S  1000    48169  46861  0  80   0 -  1091 wait   pts/0      00:00:00 sh
0 R  1000    48255  48169  0  80   0 -  9316 -      pts/0      00:00:00 ps
User Shell >> ps -l | sort
0 R  1000    48257  48256  0  80   0 -  9316 -      pts/0      00:00:00 ps
0 S  1000    46861  46856  0  80   0 -  7706 wait   pts/0      00:00:00 bash
0 S  1000    48169  46861  0  80   0 -  1091 wait   pts/0      00:00:00 sh
1 S  1000    48256  48169  0  80   0 -  1091 wait   pts/0      00:00:00 sh
F S  UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
User Shell >> ps -l | grep wait | sort
0 S  1000    46861  46856  0  80   0 -  7706 wait   pts/0      00:00:00 bash
0 S  1000    48169  46861  0  80   0 -  1091 wait   pts/0      00:00:00 sh
1 S  1000    48271  48169  0  80   0 -  1091 wait   pts/0      00:00:00 sh
User Shell >> ps -l | grep wait | sort > file1
User Shell >> cat file1
0 S  1000    46861  46856  0  80   0 -  7706 wait   pts/0      00:00:00 bash
0 S  1000    48169  46861  0  80   0 -  1091 wait   pts/0      00:00:00 sh
1 S  1000    48274  48169  0  80   0 -  1091 wait   pts/0      00:00:00 sh
User Shell >>
```

파이프라인, 다중 파이프라인, 파이프라인과 리다이렉션의 사용

## □ 백그라운드 실행 및 시그널 처리 기능 구현하기

### ■ ‘&’ 기호를 이용하여 백그라운드로 명령 실행하기 (총 3점)

#### □ ‘&’ 기호와 명령어 분리하기 (0.5점)

```
if ((backp = strchr(inbuf, '&')) != NULL)
{
    background = 1;
    *backp = 0;
}
else
    background = 0;
```

‘&’ 기호가 명령어의 끝이라 가정하고 해당 기호가 발견되면 백그라운드로 실행시키기 위해 변수를 설정해준다.

#### □ 입력받은 명령어 백그라운드로 실행하기 (1점)

#### □ 파이프를 이용한 명령을 백그라운드로 실행하기 (1.5점)

```
else if (childpid == 0)
{
    if (background && (setpgid(0, 0) == -1))
        return 1;
    if ((sigaction(SIGINT, &defhandler, NULL) == -1) ||
        (sigaction(SIGQUIT, &defhandler, NULL) == -1) ||
        (sigprocmask(SIG_UNBLOCK, &blockmask, NULL) == -1))
    {
        perror("Failed to set signal handling for command");
        return 1;
    }
    executecmd(inbuf);
    return 1;
}
if (sigprocmask(SIG_UNBLOCK, &blockmask, NULL) == -1)
    perror("Failed to unblock signals");
if (!background)
    waitpid(childpid, NULL, 0);
while (waitpid(-1, NULL, WNOHANG) > 0);
```

자식 프로세스를 만들어 백그라운드 모드로 실행되어야 한다면 setpgid를 호출하여 더 이상 포그라운드 프로세스 그룹에 속하지 않도록 한다. executecmd 함수에는 파이프 라인을 구분하는 부분이 포함되어 있으므로 파이프 명령 또한 백그라운드로 실행할 수 있다. 커맨드가 백그라운드 명령이 아니라면 해당 자식 프로세스가 종료될 때까지 기다린다. 백그라운드 명령이라면, WNOHANG 옵션을 이용한 waitpid 함수를 호출하여 백그라운드 프로세스가 시그널에 의해 종료되었는지 확인한다. 이는 백그라운드 프로세스가 좀비 프로세스가 되는 문제를 해결하기 위함이다.

```
User Shell >> sleep 5 &
User Shell >> ps
  PID TTY          TIME CMD
 46861 pts/0        00:00:00 bash
 48518 pts/0        00:00:00 sh
 48553 pts/0        00:00:00 sleep
 48556 pts/0        00:00:00 ps
User Shell >> touch file3 | sleep 5 &
User Shell >> ps
  PID TTY          TIME CMD
 46861 pts/0        00:00:00 bash
 48518 pts/0        00:00:00 sh
 48557 pts/0        00:00:00 sleep
 48560 pts/0        00:00:00 ps
User Shell >> ls
executepipe.o      file2      executepipe.c    signalsetup.c
getch.h            file3      makeargv.o       main.o
Makefile           main.c     sh               myls.o
executeredirect.o makeargv.c signalsetup.o     myls.c
executeredirect.c file1
User Shell >> █
```

## ■ 'SIGINT', 'SIGQUIT' 시그널 무시하도록 구현하기 (총 2점)

### □ 시그널 함수를 'SIGINT', 'SIGQUIT' 동작 재정의하기 (2점)

#### ■ ex) SIGINT(Ctrl + C) 시그널로 Shell 종료를 방지하기

```
int signalsetup(struct sigaction* def, sigset_t* mask, void (*handler)(int))
{
    struct sigaction catch;

    catch.sa_handler = handler;
    def->sa_handler = SIG_DFL;
    catch.sa_flags = 0;
    def->sa_flags = 0;
    if ((sigemptyset(&(def->sa_mask)) == -1) ||
        (sigemptyset(&(catch.sa_mask)) == -1) ||
        (sigaddset(&(catch.sa_mask), SIGINT) == -1) ||
        (sigaddset(&(catch.sa_mask), SIGQUIT) == -1) ||
        (sigaction(SIGINT, &catch, NULL) == -1) ||
        (sigaction(SIGQUIT, &catch, NULL) == -1) ||
        (sigemptyset(mask) == -1) ||
        (sigaddset(mask, SIGINT) == -1) ||
        (sigaddset(mask, SIGQUIT) == -1))
    {
        fprintf(stderr, "Failed to set signal handler.\n");
        return -1;
    }
    return 0;
}
```

signalsetup 함수는 시그널 구조체를 설정하는 함수이다. 인자로 받은 def엔 시그널의 디폴트 값을 저장하고 구조체 catch엔 인자로 받은 handler로 구조체를 설정한다. mask엔 SIGINT와 SIGQUIT 시그널을 추가한다.

```
if (signalsetup(&defhandler, &blockmask, SIG_IGN) == -1)
{
    perror("Failed to set up shell signal handling");
    return 1;
}
if (sigprocmask(SIG_BLOCK, &blockmask, NULL) == -1)
{
    perror("Failed to block signals");
    return 1;
}

if ((sigaction(SIGINT, &defhandler, NULL) == -1) ||
    (sigaction(SIGQUIT, &defhandler, NULL) == -1) ||
    (sigprocmask(SIG_UNBLOCK, &blockmask, NULL) == -1))
{
    perror("Failed to set signal handling for command");
    return 1;
}
```

부모 프로세스에서 SIGINT와 SIGQUIT를 무시하고 블록시킨다. 자식 프로세스에서는 명령어 실행 전 두 시그널의 핸들러를 디폴트로 재설정 후 언블록 시킨다. 이는 실행 명령 프로세스는 Ctrl-C로 종료될 수 있어야 하기 때문이다.

User Shell >>

Ctrl-C 입력으로 Shell 종료 방지



## □ 고찰

코딩을 어떻게 시작해야할지부터 막막했다. fork로 자식 프로세스를 만들어 자식 프로세스가 execvp로 명령어를 실행하면 되겠다 생각했으나, 명령어를 옵션 인자와 분리하는 것부터 문제였다. 순수히 혼자 힘으로 어려울 것 같아 인터넷과 서적을 뒤져가며 문자열 관련 함수에 대해 배우고 적용하였다. 그 외 부분들은 지금까지 배웠던 시스템소프트웨어 지식과 인터넷 및 서적 정보를 통해 하나씩 해결해 나갈 수 있었다. 탭 자동완성 기능은 인터넷에도 관련 정보가 전무하고 구현 알고리즘이 도저히 떠오르지 않아 끝내 구현하지 못하였다.

이 프로젝트는 단순히 해결로 끝나는 프로젝트가 아닌 진정으로 배울 수 있었던 프로젝트였다. 프로젝트를 진행하면서 지금까지 배웠던 내용들이 이렇게 활용될 수 있구나, 이런 함수들도 있구나 하며 많은 것을 배우고 공부할 수 있었다. 지금까지 배웠던 시스템소프트웨어 지식을 총정리하고 프로그래밍에 대해 더 배운 좋은 기회였다.