

1. 작성 함수 설명

[redirectIn]

설명

```
/*
 * redirectIn - 파일을 오픈해서 파일 내용을 표준입력으로 받는다.
 */
void redirectIn(char **fileName)
{
    int in = open(*fileName, O_RDONLY); // 파일 오픈 - 옵션 : 읽기전용
    dup2(in, STDIN_FILENO); //표준 입력 받기
    close(in); // 파일 닫기
}
```

static void cmdexec(char *cmd) 함수에서 명령어 인자 argv 중 한 인자를 받아서 해당 인자 이름을 가진 파일을 열어 내용을 표준 입력 받는 함수입니다.

1. man open : 이미 존재하는 파일을 열거나 새로운 파일 생성하는 system call

int open(const char *pathname, int flags)

flags : O_RDONLY(read_only), O_WRONLY(write_only, O_RDWR(read_write)

*출처 : <https://man7.org/linux/man-pages/man2/openat.2.html>

2. man dup2(int fd, int fd2) : 파일 식별자를 복제해 fd2를 fd로 변경합니다.

STDIN_FILENO : 표준입력 목적 파일 디스크립터입니다.

*출처 : <https://man7.org/linux/man-pages/man2/dup.2.html>

open함수로 파일을 열고 dup2함수를 통해 파일 내용을 표준입력으로 받습니다. 그후 사용을 다한 파일을 닫습니다.

[redirectOut]

```

/*
 * redirectOut - 파일을 오픈해서 파일에 표준출력을 기록한다.
 * 파일이 이미 존재할 시 파일 내용을 초기화 한다.
 * 파일이 존재하지 않을 시 파일을 새로 생성하고 권한을 부여한다.
 */
void redirectOut(char **fileName)
{
    int out = open(*fileName, O_WRONLY | O_TRUNC | O_CREAT, 0600); // 파일 오픈 - 옵션 :
    // 쓰기전용 | 파일 존재시 내부 내용 초기화 | 파일 미존재시 생성, 생성시 파일 권한
    dup2(out, STDOUT_FILENO); // 표준 출력 받기
    close(out);
}

```

static void cmdexec(char *cmd) 함수에서 명령어 인자 argv 중 한 인자를 받아서 해당 인자 이름을 가진 파일을 열어 표준 입력을 파일에 덮어씌우는 함수입니다.

1. man open의 flag

O_TRUNC : write 기능이 있는 flag가 있을시 이미 존재하는 파일을 열 경우 파일 내용을 지우고 표준 입력을 기록합니다.

O_CREAT : 파일이 존재하지 않을 시 파일을 새로 만들고 파라미터에 권한을 넣습니다.

2. STDOUT_FILENO : 표준 출력 목적 파일 디스크립터입니다.

* 출처 : redirectIn과 동일.

open함수로 파일을 열고 dup2함수를 통해 표준 입력을 파일에 기록합니다. 이때 파일이 없으면 파일을 만든 후 복제하고, 파일이 이미 존재한다면 덮어씌웁니다. 그후 사용을 다한 파일을 닫습니다.

[runPartCmd]

```

* runPartCmd - 인수로 받은 명령어를 실행한다.
*/
void runPartCmd(char **argv)
{
    execvp(argv[0], argv);
}

```

인수로 받은 명령어 배열을 실행합니다.

1. man execvp : int execvp(const char* path, char *const argv[])

path에 등록된 argv 명령어를 실행합니다.

* 출처 : <https://linux.die.net/man/3/execvp>

[createPipe]

```

/*
* createPipe - 파이프를 생성하여, 인수로 받은 명령어를 표준 입력으로 바꿔준다.
*/
void createPipe(char **argv)
{
    int fd[2];
    pid_t pid;

    if(pipe(fd) == -1) // 파이프 호풀. 2개의 fd 배열을 채운다.
    {
        exit(1); //error일 시 종료한다.
    }
    pid = fork(); //fork()
    if(pid == -1) //error일 시 종료한다.
    {
        exit(1);
    }
    else if(pid == 0) //자식프로세스
    {
        dup2(fd[1], STDOUT_FILENO); // 표준 출력을 준비한다.
        close(fd[0]);

        runPartCmd(argv); // 자식 프로세스 - 현재 파이프를 기준으로 좌측 명령어들 실행
    }
    else // 부모프로세스
    {
        wait(NULL); //아무 자식 프로세스가 종료하기를 기다린다.

        dup2(fd[0], STDIN_FILENO); // 부모 프로세스 - 현재 파이프를 기준으로 자식
        프로세스가 실행한 명령어들을 파이프 우측 명령어에 입력
        close(fd[1]);
    }
}

```

static void cmdexec(char *cmd) 함수에서 파이프 라인 "|"가 있다면 파이프라인 프로세스를 복제하여 자식 프로세스에게 이전까지의 명령어 인자 배열 argv를 실행한 결과를 표준 출력 시킨 후, 부모 프로세스에서는 자식 프로세스가 만든 표준 출력을 표준 입력으로 받습니다.

[cmdexec 수정]

```

/*

```

- * argv에 저장된 명령어 인자를 돌려 인자에 따라 명령을 실행, 기록 한다.
- * 리다이렉션의 경우 다음 인자에 대해 명령어를 실행한다.
- * 파이프의 경우 파이프 이전의 명령어를 실행 후 이후 명령어에 표준 입력한다.
- * 나머지 명령어들은 추후 명령 실행을 위해 기록한다.

```
*/
char **arg = argv;
int i = 0;
while(*arg)
{
    if(**arg == '<')
    {
        redirectIn(++arg);
    }
    else if(**arg == '>')
    {
        redirectOut(++arg);
    }
    else if(**arg == '|')
    {
        argv[i] = NULL;
        createPipe(argv);
        i = 0;
    }
    else
    {
        argv[i] = *arg;
        i++;
    }
    ++arg;
}
argv[i] = NULL;
runPartCmd(argv);
```

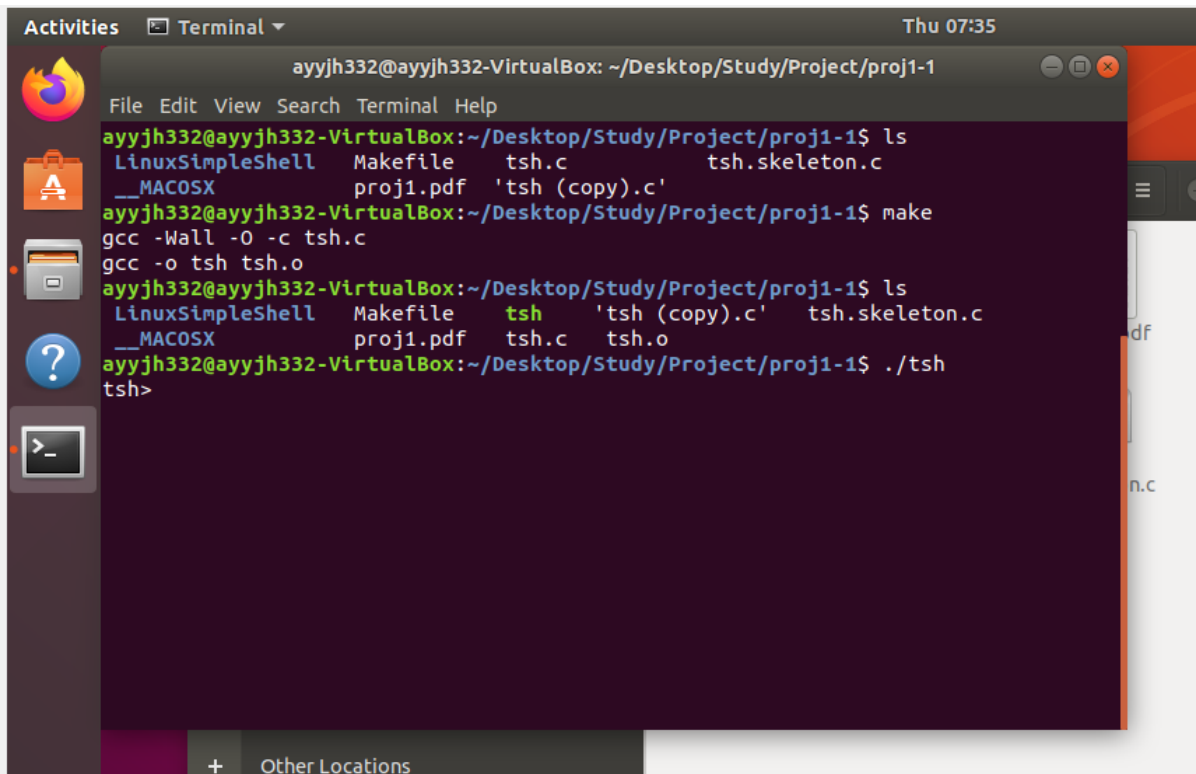
기존 단순히 인자가 1개 이상 있다면 명령을 실행하는 부분을 수정했습니다.

cmdexec에서 명령어를 각 부분별로 분리하여 argv 인자 배열을 만들었다면, 더블 포인터 arg를 이용해 분리된 인자를 하나씩 순회하며 해당 명령어에 따라 적합한 작업을 거칩니다.

1. 명령어가 "<"라면 배열에 다음 인자를 redirectIn 함수를 통해 인자 이름을 가진 파일에 내용을 표준 출력합니다.
2. 명령어가 ">"라면 배열에 다음 인자를 redirectOut을 통해 해당 인자 이름을 가진 파일에 내용을 표준 입력합니다.
3. 명령어가 "|"라면 createPipe함수를 통해 지금까지 순회한 파이프 이전까지의 명령들을 복제를 통해 자식 프로세스에게 실행시킨 후, 표준 출력을 합니다. 이후 자식 프로세스를 기다리던 부모 프로세스를 통해 표준 출력을 표준 입력으로 받아줍니다.
4. 명령어가 위 1~3에 해당하지 않는다면 해당 명령어를 기록해줍니다.
5. 배열을 순회하던 arg가 순회를 끝냈다면, 아직까지 남아있는 기록된 명령어를 실행해줌으로써 아직 실행되지 못한 명령어를 실행합니다.

* 출처 : <https://gist.github.com/tam5/be8e818d4c77dc480451> > main 함수 응용

-컴파일 과정



The screenshot shows a terminal window titled "ayyjh332@ayyjh332-VirtualBox: ~/Desktop/Study/Project/proj1-1". The terminal output is as follows:

```
ayyjh332@ayyjh332-VirtualBox:~/Desktop/Study/Project/proj1-1$ ls
LinuxSimpleShell  Makefile      tsh.c          tsh.skeleton.c
__MACOSX          proj1.pdf     'tsh (copy).c'

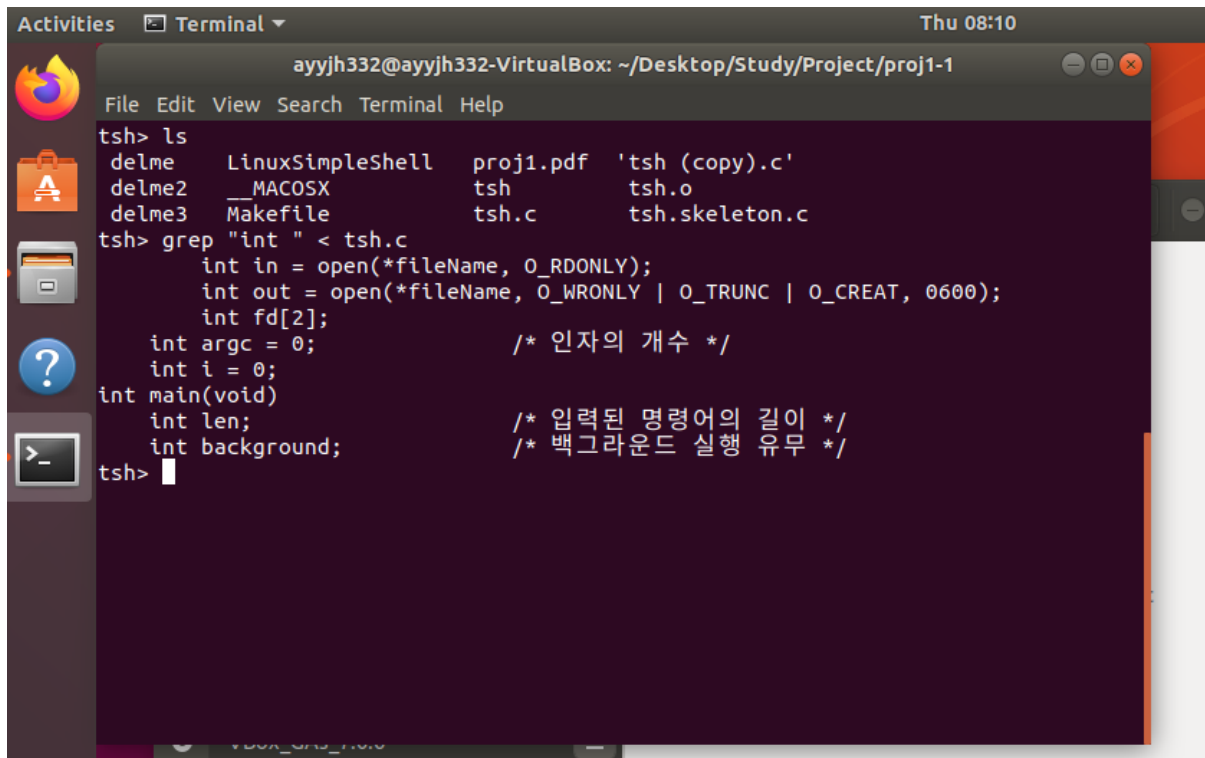
ayyjh332@ayyjh332-VirtualBox:~/Desktop/Study/Project/proj1-1$ make
gcc -Wall -O -c tsh.c
gcc -o tsh tsh.o

ayyjh332@ayyjh332-VirtualBox:~/Desktop/Study/Project/proj1-1$ ls
LinuxSimpleShell  Makefile      tsh             'tsh (copy).c'  tsh.skeleton.c
__MACOSX          proj1.pdf     tsh.c          tsh.o

ayyjh332@ayyjh332-VirtualBox:~/Desktop/Study/Project/proj1-1$ ./tsh
tsh>
```

-실행 결과물 상세 설명

검증 : 2-1



```
ayyjh332@ayyjh332-VirtualBox: ~/Desktop/Study/Project/proj1-1
File Edit View Search Terminal Help
tsh> ls
delme    LinuxSimpleShell  proj1.pdf  'tsh (copy).c'
delme2   __MACOSX          tsh        tsh.o
delme3   Makefile          tsh.c      tsh.skeleton.c
tsh> grep "int " < tsh.c
      int in = open(*fileName, O_RDONLY);
      int out = open(*fileName, O_WRONLY | O_TRUNC | O_CREAT, 0600);
      int fd[2];
      int argc = 0;                /* 인자의 개수 */
      int i = 0;
int main(void)
      int len;                    /* 입력된 명령어의 길이 */
      int background;            /* 백그라운드 실행 유무 */
tsh>
```

grep "int " < tsh.c 명령어 가공 결과 : argv = [grep, "int ", <, tsh.c]

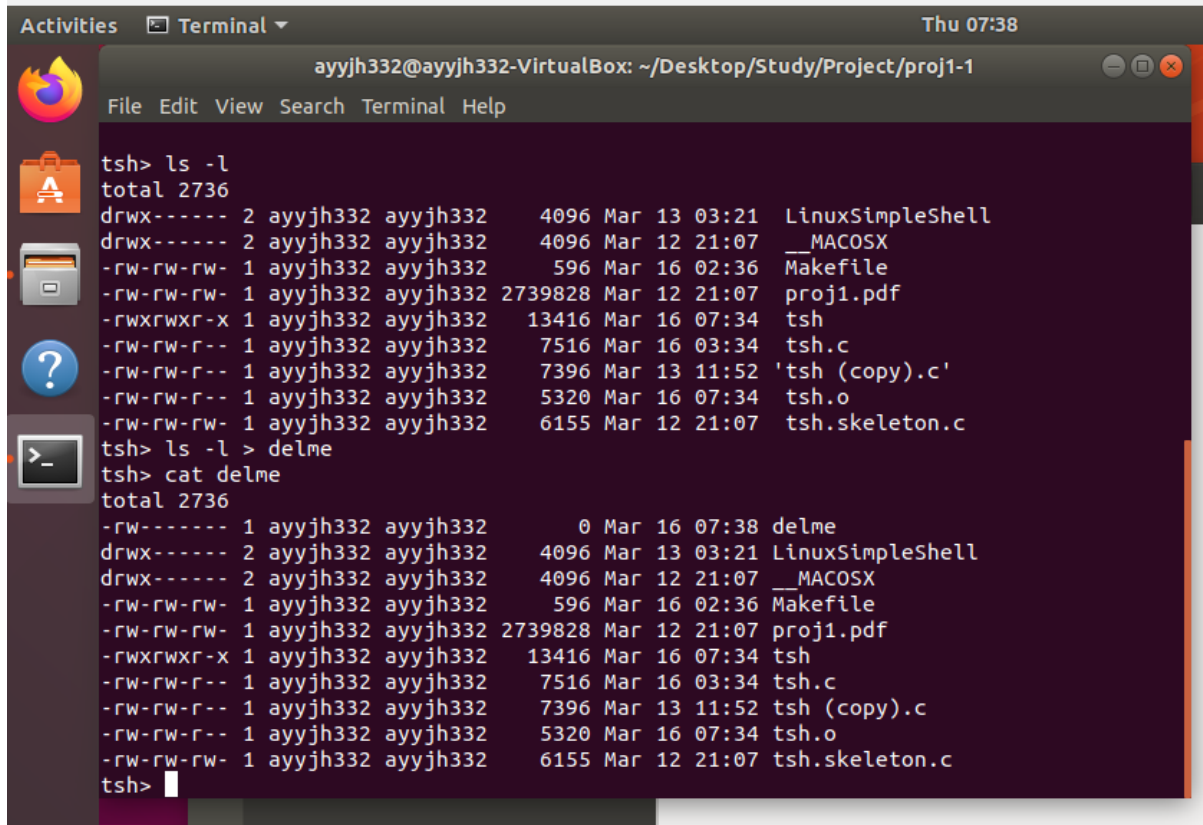
grep [옵션] [패턴] [파일명] : 특정 파일에서 지정한 문자열이나 정규표현식을 포함한 행을 출력합니다.

grep "int " < tsh.c : tsh.c 파일에 "int" 문자열이 포함된 행을 모두 출력합니다.

*참고자료 출처 : linuxcommand.org man grep -

https://linuxcommand.org/lc3_man_pages/grep1.html

검증 : 2-2 ~ 2-4



```
tsh> ls -l
total 2736
drwx----- 2 ayyjh332 ayyjh332 4096 Mar 13 03:21 LinuxSimpleShell
drwx----- 2 ayyjh332 ayyjh332 4096 Mar 12 21:07 __MACOSX
-rw-rw-rw- 1 ayyjh332 ayyjh332 596 Mar 16 02:36 Makefile
-rw-rw-rw- 1 ayyjh332 ayyjh332 2739828 Mar 12 21:07 proj1.pdf
-rwxrwxr-x 1 ayyjh332 ayyjh332 13416 Mar 16 07:34 tsh
-rw-rw-r-- 1 ayyjh332 ayyjh332 7516 Mar 16 03:34 tsh.c
-rw-rw-r-- 1 ayyjh332 ayyjh332 7396 Mar 13 11:52 'tsh (copy).c'
-rw-rw-r-- 1 ayyjh332 ayyjh332 5320 Mar 16 07:34 tsh.o
-rw-rw-rw- 1 ayyjh332 ayyjh332 6155 Mar 12 21:07 tsh.skeleton.c

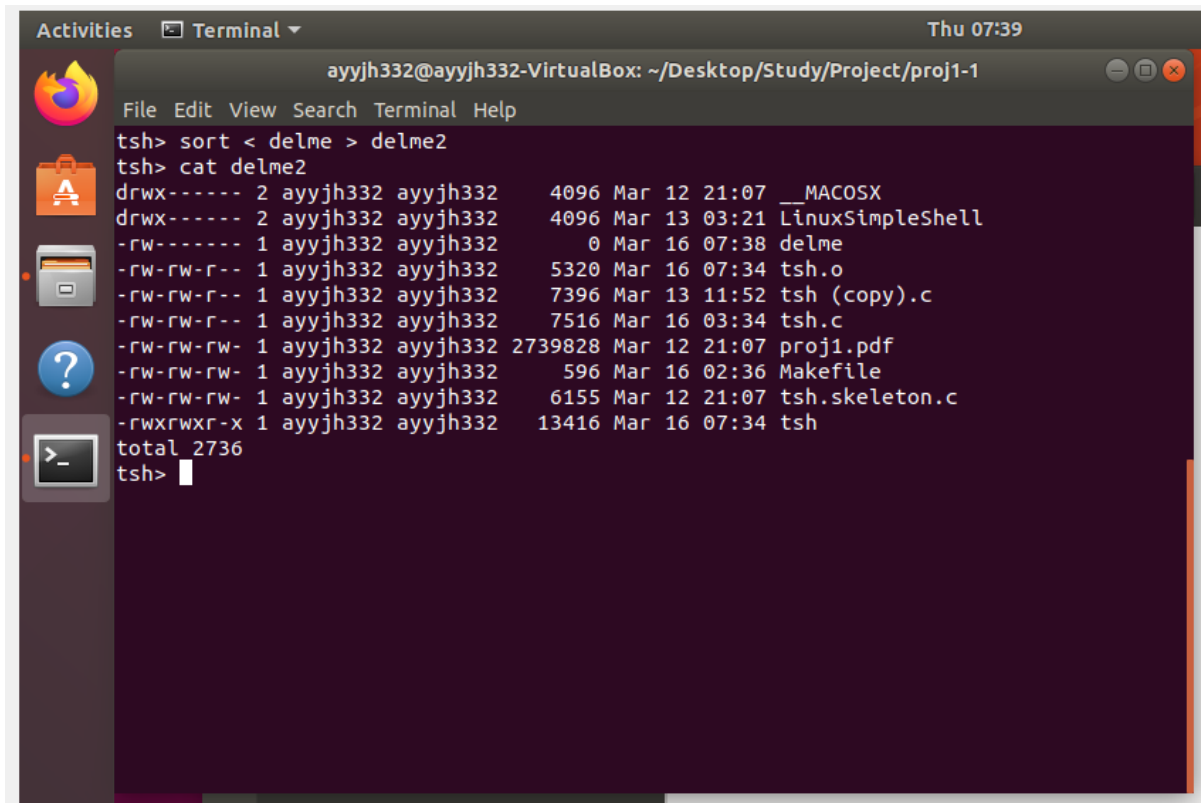
tsh> ls -l > delme
tsh> cat delme
total 2736
-rw----- 1 ayyjh332 ayyjh332 0 Mar 16 07:38 delme
drwx----- 2 ayyjh332 ayyjh332 4096 Mar 13 03:21 LinuxSimpleShell
drwx----- 2 ayyjh332 ayyjh332 4096 Mar 12 21:07 __MACOSX
-rw-rw-rw- 1 ayyjh332 ayyjh332 596 Mar 16 02:36 Makefile
-rw-rw-rw- 1 ayyjh332 ayyjh332 2739828 Mar 12 21:07 proj1.pdf
-rwxrwxr-x 1 ayyjh332 ayyjh332 13416 Mar 16 07:34 tsh
-rw-rw-r-- 1 ayyjh332 ayyjh332 7516 Mar 16 03:34 tsh.c
-rw-rw-r-- 1 ayyjh332 ayyjh332 7396 Mar 13 11:52 tsh (copy).c
-rw-rw-r-- 1 ayyjh332 ayyjh332 5320 Mar 16 07:34 tsh.o
-rw-rw-rw- 1 ayyjh332 ayyjh332 6155 Mar 12 21:07 tsh.skeleton.c
tsh>
```

ls -l > delme : 디렉토리 내 리스트를 -l 옵션으로 나온 출력을 delme 파일에 덮어씹습니다.

cat delme : delme 파일에 모든 내용을 출력합니다.

여기서 delme에 리스트에 delme가 있는 이유는 delme 파일을 먼저 만들고 나서 ls -l 명령어를 수행하기에, delme 파일이 먼저 생성된 상태입니다.

bash에서도 테스트 해본 결과 동일한 결과가 나오는 것을 보면 bash 또한 먼저 파일을 만들고 이전 명령어를 실행하는 것으로 보입니다.

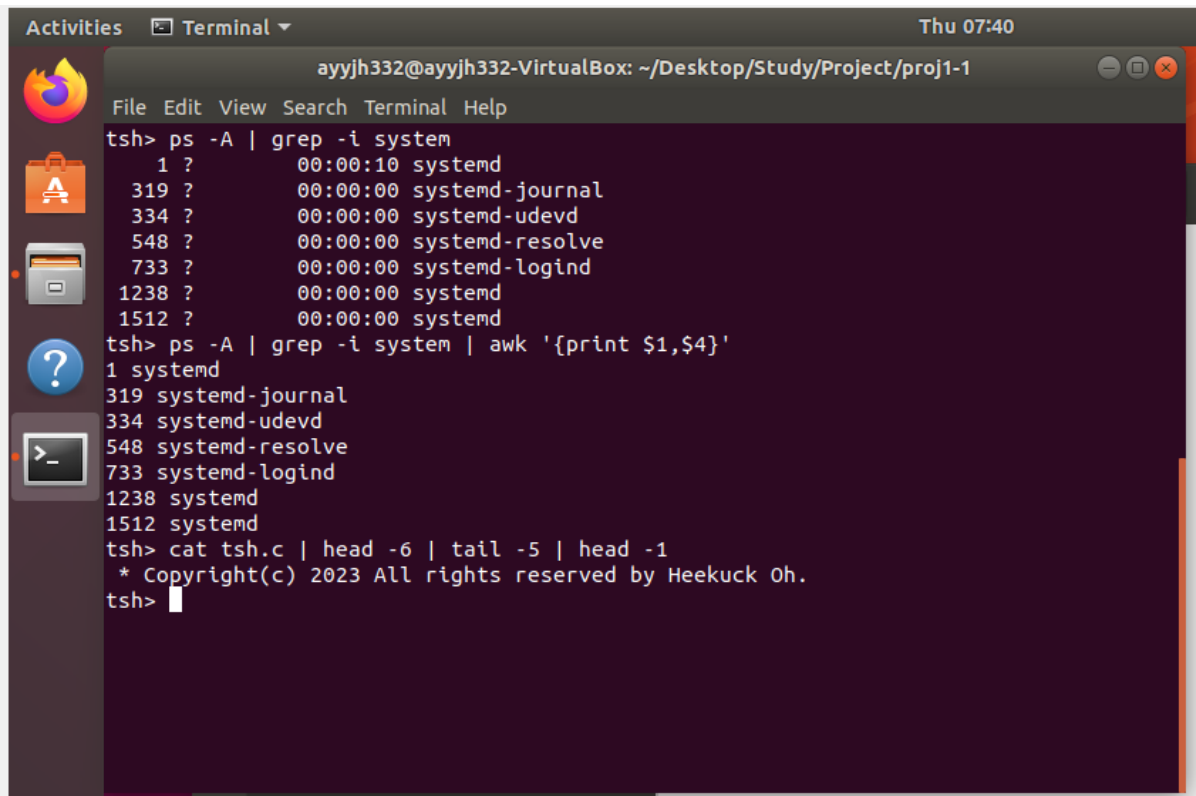


```
ayyjh332@ayyjh332-VirtualBox: ~/Desktop/Study/Project/proj1-1
tsh> sort < delme > delme2
tsh> cat delme2
drwx----- 2 ayyjh332 ayyjh332 4096 Mar 12 21:07 __MACOSX
drwx----- 2 ayyjh332 ayyjh332 4096 Mar 13 03:21 LinuxSimpleShell
-rw----- 1 ayyjh332 ayyjh332 0 Mar 16 07:38 delme
-rw-rw-r-- 1 ayyjh332 ayyjh332 5320 Mar 16 07:34 tsh.o
-rw-rw-r-- 1 ayyjh332 ayyjh332 7396 Mar 13 11:52 tsh (copy).c
-rw-rw-r-- 1 ayyjh332 ayyjh332 7516 Mar 16 03:34 tsh.c
-rw-rw-rw- 1 ayyjh332 ayyjh332 2739828 Mar 12 21:07 proj1.pdf
-rw-rw-rw- 1 ayyjh332 ayyjh332 596 Mar 16 02:36 Makefile
-rw-rw-rw- 1 ayyjh332 ayyjh332 6155 Mar 12 21:07 tsh.skeleton.c
-rwxrwxr-x 1 ayyjh332 ayyjh332 13416 Mar 16 07:34 tsh
total 2736
tsh>
```

sort < delme > delme2 : delme 파일 내용을 정렬합니다(내림차순) 이후 정렬 내용을 delme2에 저장합니다.

cat delme2 : delme2 내용을 전체 출력합니다.

*참고자료 출처 : man7.org man sort - <https://man7.org/linux/man-pages/man1/sort.1.html>

A screenshot of a Linux terminal window titled 'Terminal' with the user 'ayyjh332' and the path '~/Desktop/Study/Project/proj1-1'. The terminal shows the output of several commands. First, 'ps -A | grep -i system' lists system processes. Then, 'ps -A | grep -i system | awk '{print \$1,\$4}'' filters the output to show process IDs and names. Finally, 'cat tsh.c | head -6 | tail -5 | head -1' displays the first line of the 'tsh.c' file, which is a copyright notice.

```
ayyjh332@ayyjh332-VirtualBox: ~/Desktop/Study/Project/proj1-1
File Edit View Search Terminal Help
tsh> ps -A | grep -i system
  1 ?      00:00:10 systemd
 319 ?      00:00:00 systemd-journal
 334 ?      00:00:00 systemd-udevd
 548 ?      00:00:00 systemd-resolve
 733 ?      00:00:00 systemd-logind
1238 ?      00:00:00 systemd
1512 ?      00:00:00 systemd
tsh> ps -A | grep -i system | awk '{print $1,$4}'
1 systemd
319 systemd-journal
334 systemd-udevd
548 systemd-resolve
733 systemd-logind
1238 systemd
1512 systemd
tsh> cat tsh.c | head -6 | tail -5 | head -1
* Copyright(c) 2023 All rights reserved by Heekuck Oh.
tsh>
```

ps -A | grep -i system : 현재 실행중인 프로세스 중 system 단어가 포함된 프로세스를 출력합니다.

ps의 옵션 -A는 All로 모든 프로세스를 출력하고, grep의 -i옵션은 ignore로 대소문자 구분 없이 찾습니다.

ps -A | grep -i system | awk '{print \$1,\$4}' : 현재 실행 중인 프로세스 중 "system"이라는 단어를 포함한 프로세스를 찾아 프로세스 ID와 실행 파일명을 출력하는 명령어입니다. awk는 문자열을 가공하는 명령어이며, '{print \$1,\$4}'는 첫 번째와 네 번째 열을 출력합니다.

cat tsh.c | head -6 | tail -5 | head -1 : tsh.c 파일 내용 중 6번째 줄부터 10번째 줄까지 출력하고, 그 중 첫 번째 줄만 출력합니다. head와 tail은 각각 파일의 앞/뒤에서 몇줄을 출력할지 결정하는 명령어입니다.

*참고 자료 출처 : ChatGPT (하단 질의문)

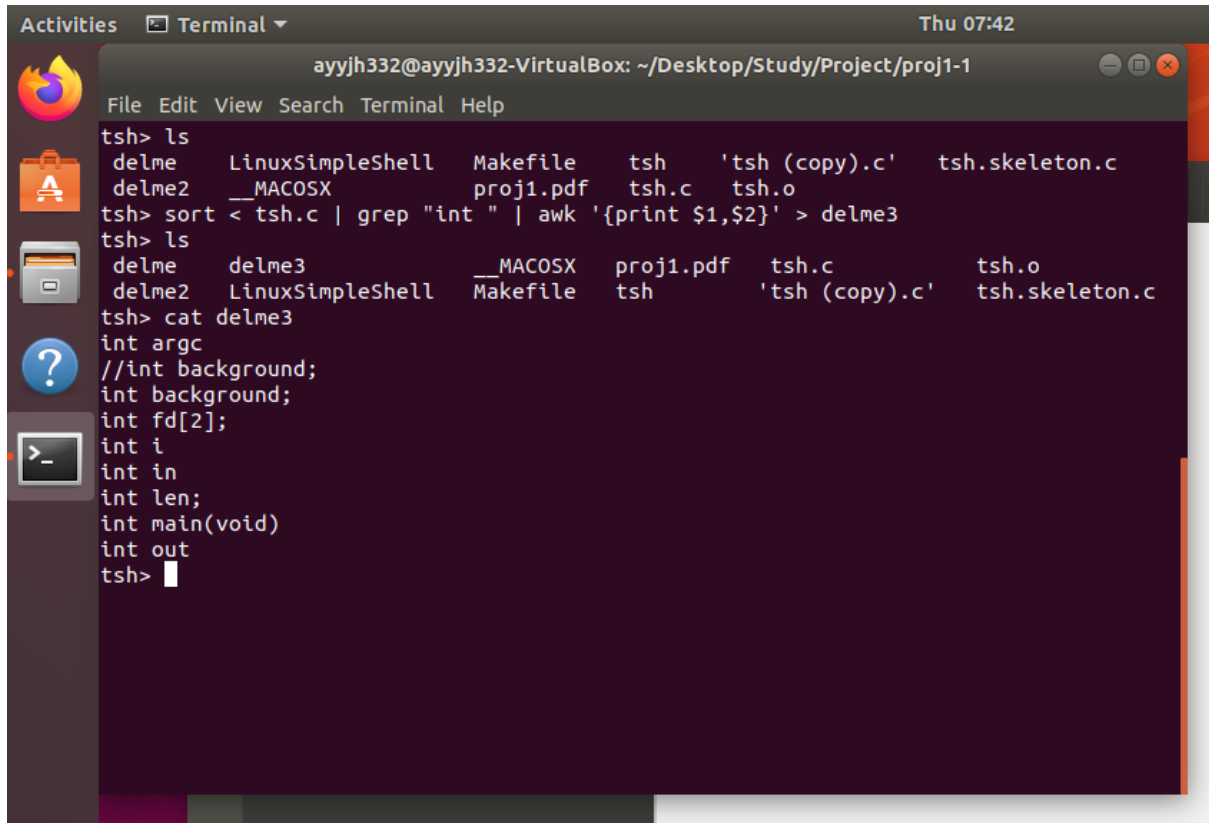
ps -A | grep -i system

ps -A | grep -i system | awk '{print \$1,\$4}'

cat tsh.c | head -6 | tail -5 | head -1 : tsh.c

위 명령들 소개할 내용 정리 해줘.

검증 : 4



```
Activities Terminal Thu 07:42
ayyjh332@ayyjh332-VirtualBox: ~/Desktop/Study/Project/proj1-1
File Edit View Search Terminal Help
tsh> ls
delme LinuxSimpleShell Makefile tsh 'tsh (copy).c' tsh.skeleton.c
delme2 __MACOSX proj1.pdf tsh.c tsh.o
tsh> sort < tsh.c | grep "int " | awk '{print $1,$2}' > delme3
tsh> ls
delme delme3 __MACOSX proj1.pdf tsh.c tsh.o
delme2 LinuxSimpleShell Makefile tsh 'tsh (copy).c' tsh.skeleton.c
tsh> cat delme3
int argc
//int background;
int background;
int fd[2];
int i
int in
int len;
int main(void)
int out
tsh>
```

sort < tsh.c | grep "int " | awk '{print \$1,\$2}' > delme3 : tsh.c 파일 내용 중 "int " 문자열을 포함하는 라인을 찾아 첫 번째 열과 두 번째 열을 출력하고, 이를 정렬한 후 delme3 파일에 저장합니다.

출처 : ChatGPT (하단 질문)

sort < tsh.c | grep "int " | awk '{print \$1,\$2}' > delme3 : tsh.c

위 명령들 소개할 내용 정리 해줘.

-과제를 수행하면서 경험한 문제점과 느낀점

과제 진행을 하면서 겪은 문제점은 파이프를 만드는 createPipe() 부분이었습니다.

createPipe()부분에 경우 초반에는 fork()를 하지 않고 바로 dup2()로 표준 입/출력을 받았으나 계속 코드가 먹통이었습니다. 이후 로그를 찍으며 코드 진행을 확인해 보니 표준 입출력이 무시되며 순서가 맞지 않는 것으로 판단되었습니다. 이후 다시 fork()로 자식 프로세스가 끝난 후 부모 프로세스에서 전달받아 순서를 맞춘 결과 정상 작동하였습니다. 이 문제상황에서 느낀점은 프로세스와 이후 배울 스레드에서 진행 흐름에 조율이 매우 중요하다고 느꼈으며, 멀티 스레드 환경에 궁금증이 더 커졌습니다.

참고 자료 출처

<https://gist.github.com/tam5/be8e818d4c77dc480451> : redirectIn, redirectOut 코드 인용,

createPipe, main 참조

<https://velog.io/@hidaehyunlee/minishell->

5. -%ED%8C%8C%EC%9D%B4%ED%94%84Pipe-%EC%B2%98%EB%A6%AC : Dup2(), Pipe 지식 참조