운영체제론 과제 #1

2021083809 컴퓨터학부 유형진

1. 작성한 코드 설명

#1) 파일 리다이렉션을 수행할 때, open()함수를 통해 파일 디스크립터를 받기 위해서 fd[]배열을 생성하였다. 여기서 파일 리다이렉션은 입력과 출력을 수행하기 때문에 크기가 2인 배열을 생성하였다.

```
int fd[2];                          // #1
```

#2) 기존코드는 문자열을 공백과 텝문자로 구분하고, 인자를 argv[]배열에 추가한다. 조건문을 추가하여서 파일 리다이렉션 기호인 '>', '<'와 파이프 기호인 '|'가 argv[]배열에 들어가지 않도록 하였다.

```
if (q == NULL || *q == ' ' || *q == '\t') {
    q = strsep(&p, " \t");
    if (*q) {
        if(*q != '<' && *q != '>' && *q != '|')   // #2
        argv[argc++] = q;
    }
}
```

#3 q가 '<'기호를 만났을 때, 실행하도록 한다. strsep()함수를 이용하여 q가 " "으로 분리된 문자열을 가리키도록 한다. 그리고 open()함수를 통해 q에 저장된 문자열(파일)을 읽기 전용 모드로 연다. 그리고 반환하는 파일 디스크립터는 fd[0]에 저장한다. 만약 open()에 오류가 생겼다면 오류메세지를 출력하고 프로그램을 종료한다. 오류가 없다면 dup2()함수를 이용하여 fd[0]을 표준 입력과 연결하여 파일에서부터 입력을 받을 수 있게 한다. 자원관리를 위해서 close()함수를 사용한다.

```
// #3
if (*q == '<') {
    q = strsep(&p, " ");
    fd[0] = open(q,O_RDONLY);
    if (fd[0] < 0) {
        perror("perror tsh.c");
        exit(EXIT_FAILURE);
    }
    dup2(fd[0], STDIN_FILENO);
    close(fd[0]);
}
```

#4 q가 '>'기호를 만났을 때, 실행하도록 한다. strsep()함수를 이용하여 q가 " "으로 분리된 문자열을 가리키도록 한다. 그리고 open()함수를 통해 q에 저장된 문자열(파일)을 파일명으로 간주하고, 읽기와 쓰기가 가능한 파일을 생성한다. 이때 만약 파일이 없다면 생성한다. 또한 모든 사용자가 읽고 쓰기 권한을 가지도록 한다. 이때 반환되는 파일 디스크립터는 fd[1]에 저장된다. 만약 open()함수가 오류가 생긴다면 오류메세지를 출력하고 프로그램을 종료한다. 오류가 없다면 dup2()함수를 통해 fd[1]과 표준 출력을 연결하여 파일로 출력을 할 수 있게 한다. 자원관리를 위해서 close()함수를 사용한다.

```
// #4
else if (*q == '>') {
    q = strsep(&p, " ");
    fd[1] = open(q,O_RDWR | O_CREAT, 0666);
    if (fd[1] < 0) {
        perror("perror delme");
        exit(EXIT_FAILURE);
    }
    dup2(fd[1], STDOUT_FILENO);
    close(fd[1]);
}
```

#5 파이프를 위한 배열 fd_p[]를 생성하였다. 입력과 출력 두가지를 수행하기 위해 배열의 크기는 2로 설정하였다.

```
int fd_p[2];                    // #5
```

#6 fork()를 하기 위해 프로세서ID pid_p를 생성한다.

```
pid_t pid_p;                    // #6
```

#7 q가 '|'기호를 만났을 때, 실행하도록 한다. pipe()함수를 호출하여 만약 문제가 생겼을 경우 오류메세지를 출력한 후 프로그램을 종료한다. fork()함수를 호출하여 자식 프로세서를 생성한다. 만약 문제가 생겼을 경우, 오류 메시지를 출력한 후 프로그램을 종료한다.

pip_p == 0, 즉 자식 프로세서일 경우 dup2()함수를 통해 명령어의 표준 출력이 파이프로 출력되어 부모 프로세서에서 파이프의 입력을 받게 한다. 그리고 argv[]배열의 끝에 NULL값을 넣어 배열의 끝을 알 수 있게 한다. 그 후 execvp()함수를 통해 argv[]배열에 저장된 명령어를 실행한다. 자원관리를 위해서 close()함수도 사용한다.

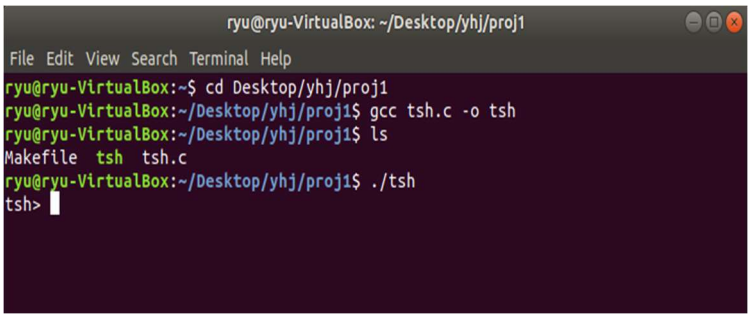
pip_p != 0, 즉 부모 프로세서일 경우 dup2()함수를 통해 명령어의 표준 입력을 파이프로 받게 한다. 그리고 재귀를 통해 남은 명령어를 실행하도록 한다. 이때 부모 프로세서는 파이프로부터 입력을 받기 때문에 자식 프로세서가 출력을 마치고 종료되면 부모 프로세서도 종료된다.

```
// #7
if (*q == '|') {
    if (pipe(fd_p) == -1) {
        printf("pipe error");
        exit(EXIT_FAILURE);
    }

    if ((pid_p = fork()) == -1) {
    perror("fork");
    exit(EXIT_FAILURE);
    }

    if (pid_p == 0) {
        close(fd_p[0]);
        dup2(fd_p[1],STDOUT_FILENO);
        close(fd_p[1]);
        argv[argc] = NULL;
        execvp(argv[0],argv);
    }
    else {
        close(fd_p[1]);
        dup2(fd_p[0], STDIN_FILENO);
        close(fd_p[0]);
        cmdexec(p);
    }
}
```

2. 컴파일 과정 및 실행 결과물

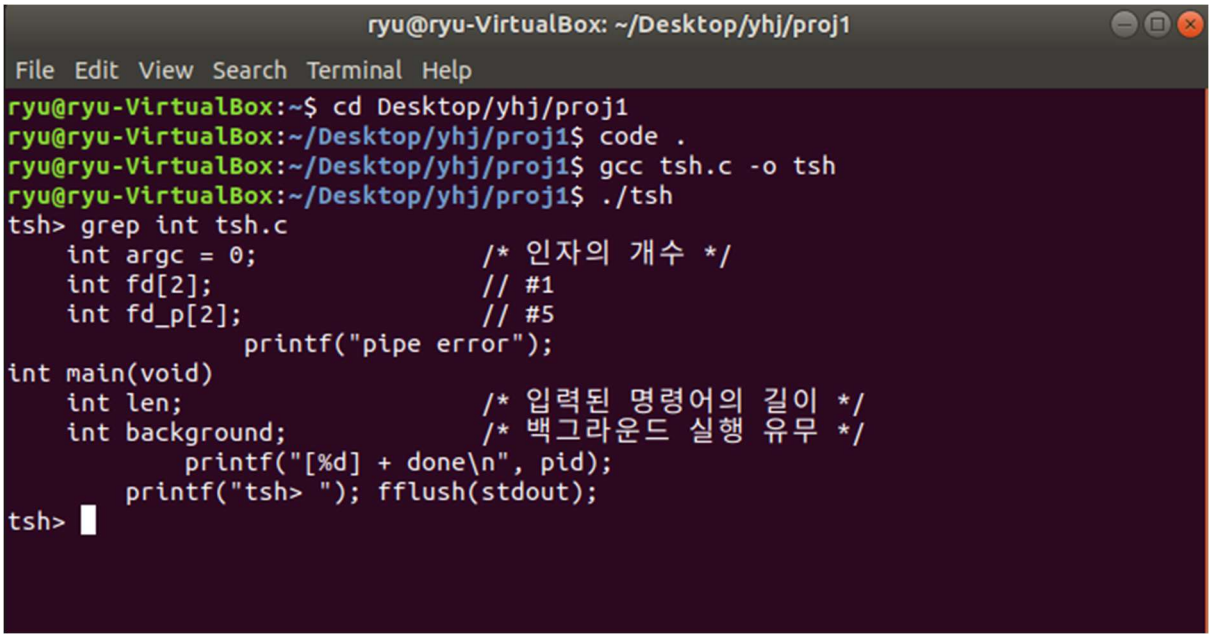
1. Terminal에서 proj1 디렉토리를 만든 후, 그곳에서

소스파일을 컴파일 한 후, 실행한다.



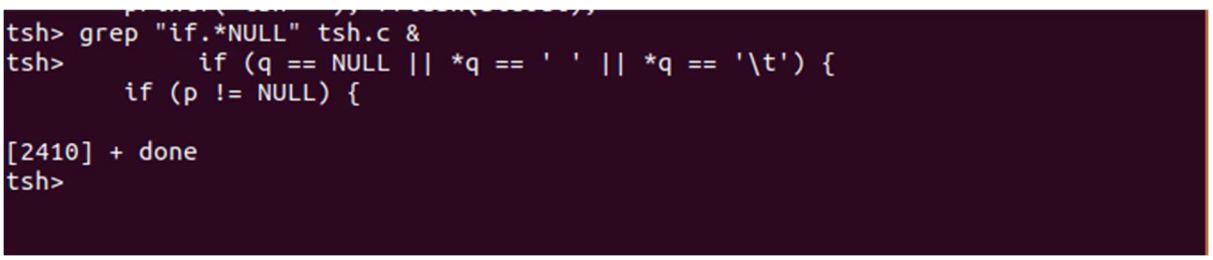
2. 명령어 grep int tsh.c 실행

- 'tsh.c' 파일에서 'int'가 포함된 모든 줄을 찾는 명령어이다.



3. 명령어 grep "if.*NULL" tsh.c & 실행

- 'tsh.c' 파일에서 "if"와 "NULL"이라는 두 단어가 등장하는 라인을 찾아 출력하는 명령어이다. 뒤에 &가 붙었기 때문에 해당 명령어는 백그라운드에서 실행된다.



4. 명령어 ps 실행

- 현재 실행중인 프로세서의 정보를 출력하는 명령어이다.

```
tsh> ps
  PID TTY          TIME CMD
 1981 pts/0    00:00:00 bash
 2310 pts/0    00:00:00 tsh
 2417 pts/0    00:00:00 ps
tsh>
```

5. 명령어 grep "int " < tsh.c 실행

- 'tsh.c'파일에서 'int'라는 문자열을 포함한 라인을 출력하는 명령어이다.

```
tsh> grep "int " < tsh.c
    int argc = 0;               /* 인자의 개수 */
    int fd[2];                  // #1
    int fd_p[2];                // #5
int main(void)
    int len;                    /* 입력된 명령어의 길이 */
    int background;             /* 백그라운드 실행 유무 */
tsh>
```

6. 명령어 ls -l 실행

- 현재 디렉토리의 파일 목록을 자세한 형식으로 출력한다. 파일 권한, 소유자, 그룹, 파일 크기, 생성 날짜 및 파일 이름을 포함한다.

```
tsh> ls -l
total 44
-rw-rw-r-- 1 ryu ryu   300  3월 30 15:03 delme
-rw-rw-r-- 1 ryu ryu   300  3월 30 15:03 delme2
-rw-rw-r-- 1 ryu ryu    73  3월 30 11:40 delme3
-rw-rw-r-- 1 ryu ryu   407  3월 22 13:52 Makefile
-rwxrwxr-x 1 ryu ryu 13224  3월 30 17:12 tsh
-rw-rw-r-- 1 ryu ryu 11268  3월 30 17:12 tsh.c
tsh>
```

7. 명령어 ls -l > delme 와 명령어 cat delme 실행

- 첫번째 명령어는 현재 디렉토리에 있는 파일과 디렉토리의 정보를 "delme"라는 이름의 파일에 덮어쓴다. 두번째 명령어는 delme의 내용을 출력한다.

```
tsh> ls -l > delme
tsh> cat delme
total 44
-rw-rw-r-- 1 ryu ryu   300  3월 30 15:03 delme
-rw-rw-r-- 1 ryu ryu   300  3월 30 15:03 delme2
-rw-rw-r-- 1 ryu ryu    73  3월 30 11:40 delme3
-rw-rw-r-- 1 ryu ryu   407  3월 22 13:52 Makefile
-rwxrwxr-x 1 ryu ryu 13224  3월 30 17:12 tsh
-rw-rw-r-- 1 ryu ryu 11268  3월 30 17:12 tsh.c
tsh>
```

8. 명령어 sort < delme > delme2 와 명령어 cat delme2 실행

- 첫번째 명령어는 "delme" 파일의 내용을 정렬한 후, 정렬된 결과를 "delme2"파일에 저장한다. 두번째 명령어는 "delme2" 파일을 출력한다.

```
tsh> sort < delme > delme2
tsh> cat delme2
-rw-rw-r-- 1 ryu ryu 11268  3월 30 17:12 tsh.c
-rw-rw-r-- 1 ryu ryu   300  3월 30 15:03 delme
-rw-rw-r-- 1 ryu ryu   300  3월 30 15:03 delme2
-rw-rw-r-- 1 ryu ryu   407  3월 22 13:52 Makefile
-rw-rw-r-- 1 ryu ryu    73  3월 30 11:40 delme3
-rwxrwxr-x 1 ryu ryu 13224  3월 30 17:12 tsh
total 44
tsh>
```

9. 명령어 ps -A | grep -i system 실행

- 현재 실행 중인 모든 프로세서를 나열하고, 그 결과에서 system이라는 문자열이 있는 프로세서를 찾아 출력한다. -i 는 대소문자를 구분하지 않는다는 뜻이다. 파이프를 이용해 ps와 grep명령어를 연결하여 사용한다.

```
tsh> ps -A | grep -i system
    1 ?        00:00:07 systemd
  281 ?        00:00:00 systemd-journal
  297 ?        00:00:00 systemd-udevd
  643 ?        00:00:00 systemd-resolve
  754 ?        00:00:00 systemd-logind
 1180 ?        00:00:00 systemd
 1402 ?        00:00:00 systemd
tsh>
```

10. 명령어 ps -A | grep -i system | awk '{print $1,$4}' 실행

- 실행 중인 모든 프로세서들을 보여주고, 그 중에서 'system'이라는 문자열이 포함된 프로세스를 찾는다.(대소문자 구분없이) 그리고 다시 그 결과를 awk 명령어로 보내어 첫번째 열과 네번째 열만을 출력한다.

```
tsh> ps -A | grep -i system | awk '{print $1,$4}'
1 systemd
281 systemd-journal
297 systemd-udevd
643 systemd-resolve
754 systemd-logind
1180 systemd
1402 systemd
tsh>
```

11. 명령어 cat tsh.c | head -6 | tail -5 | head -1 실행

- tsh.c 파일의 내용을 출력하고, 그 중에서 첫 6줄을 출력한 후, 그 중에서 마지막 5줄을 출력한 다음, 그 중 첫번째 줄을 출력한다.

```
tsh> cat tsh.c | head -6 | tail -5 | head -1
 * Copyright(c) 2023 All rights reserved by Heekuck Oh.
tsh>
```

12. 명령어 sort < tsh.c | grep “int ” | awk ‘{print $1,$2}’ > delme3 와 cat delme3 실행

- 첫번째 명령어는 ‘tsh.c’ 파일을 정렬하고, 각 줄에서 “int”라는 문자열을 찾아 첫 번째와 두 번째 단어를 출력한 후, 이를 ‘delme3’ 파일에 저장한다. 두번째 명령어는 ‘delme3’ 파일을 출력한다.

```
tsh> sort < tsh.c | grep "int " | awk '{print $1,$2}' > delme3
tsh> cat delme3
int argc
int background;
int fd[2];
int fd_p[2];
int len;
int main(void)
tsh>
```

3. 과제를 수행하면서 경험한 문제점과 느낀 점

1) 과제를 늦게 시작해서 마음이 급한 나머지 코드를 처음부터 보고 이해한 다음에 코딩을 한게 아니라 그냥 냅다 코딩부터 시작하였다. 당연히 속도가 나지 않았고 결국 뼈대코드를 전부 이해한 뒤에야 속도가 붙었다. 앞으로도 과제를 할 때, 뼈대가 주어진다면 일단 그것을 완벽히 이해한 후 과제를 시작하는 것이 속도도 빠르고 얻어가는 것 많을 것이라고 생각한다.

2) 초반 파일 리다이렉션 구현중 발생한 문제이다. 실행결과물에만 집착하여 open()함수에 tsh.c라는 파일명을 그대로 넣었었다. 파일 리다이렉션을 다 구현할 때까지 인지를 못하다가 코드를 한번 처음부터 살펴보던 중에 발견하였다. 너무 결과에만 집착하지 말고 이 과제에서 얻어갈 수 있는 것과 요구하는 것을 생각하면서 과제를 해결해야 함을 다시 한번 깨달았다.

3) 파이프를 구현하면서 발생한 문제이다. 시작할 때는 파이프에 대한 이해와 프로세서의 대한 이해, 프로세서간 통신을 잘 이해하고 있다고 생각했었다. 하지만 파이프와 다중파이프를 구현하면서 내가 잘못 알고 있는 것이 많다는 것을 깨달았다. 예를 들어 나는 cmdexec()를 돌려야 하는 이유를 몰랐었다. 자식 프로세서에서 execvp()를 하면 바로 부모 프로세서로 넘어간다고 착각을 했다. 하지만 이것은 나의 착각이었고, 교수님이 카톡방에 올려 주신 내용을 보고 다시 생각한 다음에 구현하게 되었다. 이번 과제를 통해 프로세스 간의 통신을 확실히 깊게 이해하게 된 것 같다.

느낀점: 본래 코딩에 딱히 재미를 느낀 적도 흥미를 느낀 적도 없었다. 아마도 이번 과제가 나에게 있어서 지금까지의 과제 중에서 제일 시간이 오래 걸리고 어려웠던 과제일 것이다. 하지만 그것이 나에게 좋지 않은 영향을 끼치지는 않았다. 오히려 좋은 영향을 받았다는 것이 맞는 말일

것 같다. 이번 과제를 성공했을 때, 지금까지 코딩에서 느껴보지 못한 기분을 느꼈다. 주변 친구들이 코딩할 때의 쾌감을 말할 때, 전혀 공감하지 못하였었는데 이번 과제를 통해 조금은 그 감정을 알 수 있게 되었다. 또한 이 과제를 통해 운영체제에 대해서 더 알고 싶었고, 운영체제에 대한 흥미가 생기게 된 계기가 되었다.