시스템프로그래밍 과제(HW4)

32191597 박민규

2022-12-03

과제 개요 : 이번 과제는 수업 시간에 배운 system call 들을 활용하여 (fork(), wait() 등) 나만의 linux shell 을 만드는 것이다. 개인적으로 고민도 많이 하고 특히 redirection 부분의 구현이 까다로웠던 것 같다. 그래도 과제 기간이 넉넉하기도 했고 수업시간에 배운 내용들을 생각하면서 작업을 하니 다행히도 요구한 사항을 모두 구현한 것 같다.

코드 설명 : 과제 디렉토리 안의 파일들로는 myshell.h main.c redirection.c tokenize.c executes.c checking.c background.c Makefile 이 있다.

"myshell.h"

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdlib.h>
#define MAXARG 7

void doRedirection(int flag, int is_bg, char *argv[], char* input, char * output);
int checkInput(char buf[]);
int tokenized(char buff[], char* arg[] ,char delim[]);
void executesLine(int isbg, char * argv[]);
```

코드 파일이 여러 개로 나눠지다 보니 헤더파일을 하나 만들어서 관리 하는게 훨씬 더 깔끔하겠다는 생각이 들어서 주요 헤더파일과 main 함수에서 쓰일 함수 원형들을 myshell.h 파일 안에 정의했다.

"checking.c"

```c
#include "myshell.h"

// 사용자에게 받는 input의 예외를 checking해주는 함수
int checkInput(char buf[]){
    if(strcmp(buf, "")!=0 && strcmp(buf, "\t")!=0 && strcmp(buf, " ")!=0)
        return 0;
    return 1;
}
```

myshell 에서 사용자에게 input 을 받을 때의 예외를 처리해주기 위한 함수인 checkInput 함수가 있다. 입력이 없거나 탭 이거나 공백일 때 return1 을 해주고 나머지 정상적인 입력이 들어왔을 때 return 0 을 해준다.

“tokenize.c”

```c
#include "myshell.h"

int tokenized(char buff[], char* arg[] ,char delim[]){
    char* saveptr;
    char* s;
    int num = 0;

    s = strtok_r(buff, delim, &saveptr);
    while(s) {
        arg[num++] = s;
        s = strtok_r(NULL, delim, &saveptr);
    }

    arg[num] = (char *)0;

    return num;
}
```

tokenize.c 파일 안에 있는 tokenized 함수는 사용자에게 입력을 받은 버퍼를 strtok_r 함수를 이용해 토큰화 해주어 파라미터로 받은 arg 에 차례대로 저장한 후, 토큰으로 저장된 값의 개수를 return 한다. 여기서 strtok 를 사용하지 않고 strtok_r 함수를 사용한 이유은 multi-threading 환경에서 사용이 가능하기 때문이다.

“executes.c”

```c
#include "myshell.h"

void executesLine(int isbg, char *argv[]){
    int status, pid;

    if ((pid=fork()) == -1)
        perror("fork failed");
    else if (pid != 0) {
        if(isbg==0)
            pid = wait(&status);
        else {
            printf("[1] %d\n", getpid());
            waitpid(pid, &status, WNOHANG);
        }
    } else {
        execvp(argv[0], argv);
    }
}
```

executes.c 파일 안의 executesLine 함수는 사용자에게 받아온 버퍼를 실제 실행하는 함수이다. 첫 번째 매개변수인 isbg 은 백그라운드 작업을 할 것이지 하지 않을 것인지를 결정하는 flag 역할이다. 우선 fork 를 통해 자식 프로세스를 생성한다. 그 후 부모 프로세스이고, 백그라운드 실행이 아니라면 wait 을 통해 자식 프로세스가 끝나는 것을 기다리고, 백그라운드 실행이라면, waitpid 함수의 세 번째 매개변수를 WNOHANG 으로 설정하여 자식 프로세스가 끝나는 것을

기다리지 않고 백그라운드 실행이 가능하도록 하였다. 그리고 자식 프로세스라면, execvp 함수를 사용하여 사용자의 명령을 그대로 실행하도록 했다.

"redirecion.c"

```c
#include "myshell.h"

void doRedirection(int flag, int is_bg, char *argv[], char* input, char * output){
    int input_fd, output_fd;
    int status, pid;

    if ((pid=fork()) == -1)
        perror("fork failed");
    else if (pid != 0) {
        if(is_bg==0)
            pid = wait(&status);
        else {
            printf("[1] %d\n", getpid());
            waitpid(pid, &status, WNOHANG);
        }
    } else {
        if (flag == 2) {
            if((input_fd = open(input, O_RDONLY))==-1){
                perror(argv[0]);
                exit(2);
            }
            dup2(input_fd, 0);
            close(input_fd);
            execvp(argv[0], argv);
        } else if (flag == 3) {
            output_fd = open(output, O_CREAT|O_TRUNC|O_WRONLY, 0600);
            dup2(output_fd, 1);
            close(output_fd);
            execvp(argv[0], argv);
        }
    }
}
```

redirection.c 파일의 doRedirection 함수는 사용자에게 redirection 명령을 받았을 때, 이를 실행하기 위한 함수이다. 이 함수를 작성하는 과정에서의 시간이 가장 많이 걸렸고, 코드도 깔끔하지 못하지만 구현에 성공했다는 것에 의의를 두었다. 첫 번째 매개변수인 flag 는 > 명령을 실행할 것인지 < 명령을 실행할 것인지에 대한 구별을 위한 것이다. Is_bg 는 앞에서 봤듯이 백그라운드 실행에 대한 매개변수이고, argv 는 사용자 입력, input 과 output 은 대상 파일 혹은 명령어를 뜻한다. 앞의 execute 코드에서 본 코드가 앞에 있고, 그 후에 flag==2 라면 < 명령을 실행하기 위해 input_fd 를 생성하고 dup2 함수를 통해 standard input 으로 복제한다. flag==3 이라면 > 명령을 실행하기 위해 output_fd 를 생성하고 dup2 함수를 통해 standard output 으로 복제한다. 구현을 하고 보니 어렵지 않았지만, 이 아이디어를 얻기까지의 시간이 많이 걸렸던 것 같다. 이제 main 함수를 실행하기 위한 함수는 모두 살펴 보았으니, main 함수를 살펴본다.

“main.c”

사실 shell 을 구성하기 위한 주요 logic 들은 위의 함수에 거의 다 있고 main 함수의 역할은 위의 함수들이 받은 매개변수를 설정해 주고 함수를 호출하는 역할 뿐이다.

```
while (1) {
    printf("myshell$ ");
    gets(buffer);

    if (buffer != NULL && checkInput(buffer)==0) {
        //multiple instruction (pipeline)
        num_args =0;
        if (strchr(buffer, '|') != NULL) {
            num_args = tokenized(buffer, args, "|");
        } else {
            num_args = 1;
            args[0] = buffer;
            args[1] = (char *)0;
        }

        if (!strcmp(args[0], "quit")) {
            printf("Goodbye from myshell::\n");
            break;
        }
```

먼저 무한루프를 돌며 사용자에게 계속 입력을 받는다. 그 후 pipline 처리를 위해 '|'가 있을 경우 이를 기준으로 strchr 함수를 적용하고 명령어의 총 개수를 return 받는다. 없을 경우 args[0]에 명령어를 그대로 저장한다. quit 이라는 명령어가 들어올 경우, 무한루프를 탈출하여 myshell 의 실행을 끝내게 된다.

```
for (i = 0; i<num_args; i++) {
    num = 0;
    m = 0;
    is_bg = 0;
    s = strtok_r(args[i], delim, &saveptr);

    while(s) {
        if (strcmp(s, "<") == 0) {
            m = 2;
        } else if (strcmp(s, ">") == 0) {
                m = 3;
        } else if(strcmp(s, "&") == 0) {
            is_bg = 1;
        } else {
            if(m < 2)
                arg[num++] = s;
            else if(m == 2)
                strcpy(input, s);
            else
                strcpy(output, s);
        }

        s = strtok_r(NULL, delim, &saveptr);

    }
    arg[num] = (char *)0;
```

그 후 명령어 개수만큼 for 문을 돌리는데, 그 안에서 공백을 기준으로 다시 strtok_r 함수를 써서 인자를 나눠주고 그 인자가 null 이 아닐 때까지 while 문을 돌린다. 그 아래는 redirection 과 background 작업을 처리해주기 위해 flag 값을 설정해주는 코드이다. < 가 들어가면 m=2, > 가 들어가면 m=3, & 가 들어가면 is_bg=1 을 넣어준다.

```
if (m == 0) {
    executesLine(is_bg, arg);
}
else if(m >= 2) {
    doRedirection(m, is_bg, arg, input, output);
}
```

그 후 설정된 flag 값에 따라 함수를 실행시키면 된다.

코드 실행 : 소스코드 파일이 여러 개로 나누어져 있어서, Makefile 을 만들어 컴파일을 쉽게 하도록 하였다.

```
CC          = gcc
CFLAGS      = -g -O2
TARGET      = myshell
OBJECTS     = checking.o executes.o redirection.o tokenize.o
MAIN_SOURCE = main.c

all: $(TARGET)

$(TARGET): $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $(OBJECTS) $(MAIN_SOURCE)

.c.o:
    $(CC) $(CFLAGS) -c $<

clean:
    rm -f $(OBJECTS)
    rm -f $(TARGET)

.PHONY:
    clean
```

make all 명령어를 터미널에 입력하면 자동으로 컴파일 후 실행파일을 만들어 주고, make clean 명령어를 입력하면 모든 오브젝트 파일과 실행파일을 삭제해준다. 실행은 기본적인 터미널 명령어과 background, redirection 명령어 실행이 정상적으로 되는지를 검사한다.

```
root@goorm:/workspace/SystemProgramming/realmyshell# make all
gcc -g -O2 -c checking.c
gcc -g -O2 -c executes.c
gcc -g -O2 -c redirection.c
gcc -g -O2 -c tokenize.c
gcc -g -O2 -o myshell checking.o executes.o redirection.o tokenize.o main.c
main.c: In function `main':
main.c:27: warning: `gets' is deprecated (declared at /usr/include/stdio.h:577)
/tmp/ccicoHoM.o: In function `main':
/workspace/SystemProgramming/realmyshell/main.c:27: warning: the `gets' function is dangerous and should not be used
.
root@goorm:/workspace/SystemProgramming/realmyshell# ./myshell
myshell$
```

먼저 make all 명령어를 통해 실행파일을 생성하고, 이를 실행한 모습이다. gets 함수를 쓰지 말라는 경고가 나오지만 본 과제에서는 무시하고 진행한다.

```
myshell$ ls
Makefile      checking.c  executes.c  main.c   myshell.h      redirection.o  tokenize.o
background.c  checking.o  executes.o  myshell  redirection.c  tokenize.c
myshell$ ls -l
합계 88
-rw-rw-r-- 1 root root   303 12월  3 12:05 Makefile
-rw-rw-r-- 1 root root   137 12월  3 11:33 background.c
-rw-rw-r-- 1 root root   295 12월  3 10:51 checking.c
-rw-rw-r-- 1 root root  6320 12월  3 12:13 checking.o
-rw-rw-r-- 1 root root   461 12월  3 11:09 executes.c
-rw-rw-r-- 1 root root  7080 12월  3 12:13 executes.o
-rw-rw-r-- 1 root root  2206 12월  3 12:04 main.c
-rwxrwxr-x 1 root root 23176 12월  3 12:13 myshell
-rw-rw-r-- 1 root root   415 12월  3 10:32 myshell.h
-rw-rw-r-- 1 root root   954 12월  3 11:33 redirection.c
-rw-rw-r-- 1 root root  7680 12월  3 12:13 redirection.o
-rw-rw-r-- 1 root root   383 12월  3 10:30 tokenize.c
-rw-rw-r-- 1 root root  6520 12월  3 12:13 tokenize.o
myshell$ cat myshell.h
/*
        이름 : 박민규
        학번 : 32191597
        날짜 : 2022-12-03
*/

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <stdlib.h>
#define MAXARG 7

void doRedirection(int flag, int is_bg, char *argv[], char* input, char * output);
int checkInput(char buf[]);
int tokenized(char buff[], char* arg[] ,char delim[]);
void executesLine(int isbg, char * argv[]);
myshell$
```

기본 터미널 명령어들이 잘 작동되는 것을 확인할 수 있다.

```
myshell$ ls
Makefile      checking.c  executes.c  main.c   myshell.h      redirection.o  tokenize.o
background.c  checking.o  executes.o  myshell  redirection.c  tokenize.c
myshell$ ls -l > test.txt
myshell$ ls
Makefile      checking.c  executes.c  main.c   myshell.h      redirection.o  tokenize.c
background.c  checking.o  executes.o  myshell  redirection.c  test.txt       tokenize.o
myshell$ cat test.txt
합계 88
-rw-rw-r-- 1 root root   303 12월  3 12:05 Makefile
-rw-rw-r-- 1 root root   137 12월  3 11:33 background.c
-rw-rw-r-- 1 root root   295 12월  3 10:51 checking.c
-rw-rw-r-- 1 root root  6320 12월  3 12:13 checking.o
-rw-rw-r-- 1 root root   461 12월  3 11:09 executes.c
-rw-rw-r-- 1 root root  7080 12월  3 12:13 executes.o
-rw-rw-r-- 1 root root  2206 12월  3 12:04 main.c
-rwxrwxr-x 1 root root 23176 12월  3 12:13 myshell
-rw-rw-r-- 1 root root   415 12월  3 10:32 myshell.h
-rw-rw-r-- 1 root root   954 12월  3 11:33 redirection.c
-rw-rw-r-- 1 root root  7680 12월  3 12:13 redirection.o
-rw------- 1 root root     0 12월  3 12:16 test.txt
-rw-rw-r-- 1 root root   383 12월  3 10:30 tokenize.c
-rw-rw-r-- 1 root root  6520 12월  3 12:13 tokenize.o
myshell$
```

ls -l 의 결과를 test.txt 파일에 redirection 하는 예제 코드이다. cat 명령어를 통해 확인한 결과 제대로 값이 들어갔음을 확인할 수 있다.

```
myshell$ ls
Makefile      checking.c  executes.c  main.c   myshell.h      redirection.o  tokenize.c
background.c  checking.o  executes.o  myshell  redirection.c  test.txt       tokenize.o
myshell$ cat < test.txt
합계 88
-rw-rw-r-- 1 root root   303 12월  3 12:05 Makefile
-rw-rw-r-- 1 root root   137 12월  3 11:33 background.c
-rw-rw-r-- 1 root root   295 12월  3 10:51 checking.c
-rw-rw-r-- 1 root root  6320 12월  3 12:13 checking.o
-rw-rw-r-- 1 root root   461 12월  3 11:09 executes.c
-rw-rw-r-- 1 root root  7080 12월  3 12:13 executes.o
-rw-rw-r-- 1 root root  2206 12월  3 12:04 main.c
-rwxrwxr-x 1 root root 23176 12월  3 12:13 myshell
-rw-rw-r-- 1 root root   415 12월  3 10:32 myshell.h
-rw-rw-r-- 1 root root   954 12월  3 11:33 redirection.c
-rw-rw-r-- 1 root root  7680 12월  3 12:13 redirection.o
-rw------- 1 root root     0 12월  3 12:16 test.txt
-rw-rw-r-- 1 root root   383 12월  3 10:30 tokenize.c
-rw-rw-r-- 1 root root  6520 12월  3 12:13 tokenize.o
myshell$
```

반대로 test.txt 의 값을 input 으로 redirection 받는 예제 코드이다 마찬가지로 아까 들어갔던 내용이 잘 출력되는 것을 볼 수 있다.

```c
/*
    이름 : 박민규
    학번 : 32191597
    날짜 : 2022-12-03
*/

#include <stdio.h>

int main(void)
{
    while(1)
    {
        ;
    }
    return 0;
}
```

백그라운드 명령이 잘 실행되는지 보기 위해 backgroun.c 파일을 만들어 무한루프를 돌린다. 백그라운드 실행이 잘 된다면 이 무한루프가 도는것과 상관 없이 터미널에 다른 명령어가 정상적으로 실행되어야 할 것이다.

```
myshell$ ls
Makefile      checking.c  executes.c  main.c   myshell.h      redirection.o  tokenize.c
background.c  checking.o  executes.o  myshell  redirection.c  test.txt       tokenize.o
myshell$ gcc -o background background.c
myshell$ ls
Makefile    background.c  checking.o  executes.o  myshell    redirection.c  test.txt    tokenize.o
background  checking.c    executes.c  main.c      myshell.h  redirection.o  tokenize.c
myshell$ ./background &
[1] 4710
myshell$ ps
    PID TTY          TIME CMD
    319 pts/4    00:00:00 bash
   4710 pts/4    00:00:00 myshell
   4732 pts/4    00:00:00 background
   4740 pts/4    00:00:00 ps
myshell$ kill -9 4732
myshell$ ps
myshell$     PID TTY          TIME CMD
    319 pts/4    00:00:00 bash
   4710 pts/4    00:00:00 myshell
   4748 pts/4    00:00:00 ps
```

background.c 를 컴파일 후 실행시키고, ps 명령어를 통해 백그라운드 작업이 되고 있는지 확인한다. 그 후 kill 명령어를 통해 해당 프로세스를 종료 시킨다. 확인 결과 백그라운드 작업도 잘 동작되는 것을 볼 수 있다.

```
myshell$ cat background.c | ls -l
/*
        이름 : 박민규
        학번 : 32191597
        날짜 : 2022-12-03
*/

#include <stdio.h>

int main(void)
{
        while(1)
        {
                ;
        }
        return 0;
}
합계 104
-rw-rw-r-- 1 root root   303 12월  3 12:05 Makefile
-rwxrwxr-x 1 root root  8224 12월  3 12:23 background
-rw-rw-r-- 1 root root   137 12월  3 11:33 background.c
-rw-rw-r-- 1 root root   295 12월  3 10:51 checking.c
-rw-rw-r-- 1 root root  6320 12월  3 12:13 checking.o
-rw-rw-r-- 1 root root   461 12월  3 11:09 executes.c
-rw-rw-r-- 1 root root  7080 12월  3 12:13 executes.o
-rw-rw-r-- 1 root root  2206 12월  3 12:04 main.c
-rwxrwxr-x 1 root root 23176 12월  3 12:13 myshell
-rw-rw-r-- 1 root root   415 12월  3 10:32 myshell.h
-rw-rw-r-- 1 root root   954 12월  3 11:33 redirection.c
-rw-rw-r-- 1 root root  7680 12월  3 12:13 redirection.o
-rw------- 1 root root   776 12월  3 12:16 test.txt
-rw-rw-r-- 1 root root   383 12월  3 10:30 tokenize.c
-rw-rw-r-- 1 root root  6520 12월  3 12:13 tokenize.o
myshell$
```

마지막으로 파이프라인 명령어를 통해 파이프라인이 잘 동작 하는지도 검사를 완료했다.

```
myshell$ quit
Goodbye from myshell::
root@goorm:/workspace/SystemProgramming/realmyshell# ls
Makefile    background.c  checking.o  executes.o  myshell    redirection.c  test.txt    tokenize.o
background  checking.c    executes.c  main.c      myshell.h  redirection.o  tokenize.c
root@goorm:/workspace/SystemProgramming/realmyshell# make clean
rm -f checking.o executes.o redirection.o tokenize.o
rm -f myshell
root@goorm:/workspace/SystemProgramming/realmyshell# ls
Makefile  background  background.c  checking.c  executes.c  main.c  myshell.h  redirection.c  test.txt  tokenize.c
root@goorm:/workspace/SystemProgramming/realmyshell#
```

myshell 을 종료시키고, make clean 명령어를 사용하며 코드 실행은 마치도록 한다.

마치며

이번 과제를 통해 수업시간에 배운 system call 과 여러가지 함수들, 부모 프로세스와 자식 프로세스와의 관계를 완벽히 이해한 것 같다. 수업시간에 들었을 때는 워낙 생소한 개념이다 보니 와닿는 느낌이 없었는데, 확실히 직접 코드를 짜보고 system call 들을 실제로 사용을 해보니 개념이 확실히 머리에 잡히는 느낌이었다. 뿐만 아니라 여러가지 파일에 코드를 나눠 작성하여 Makefile 까지 사용해서, 완성도 있는 코드를 작성한 것 같다. 코드의 디테일 한 부분에서 좀 더 깔끔하게 짤 수 있지 않을까 하는 아쉬움도 남지만, 아쉬움 보다는 성취감이 더 큰 과제였던 것 같다.