[운영체제론]
프로젝트 1 : 미니 쉘 만들기

2017012642 강동진

알고리즘

fgets를 통해 명령어를 입력받는다.

입력받은 명령어를 strtok를 통해 공백 단위로 split하여 캐릭터 포인터 배열에 삽입한다.

fgets로 받은 문자열은 개행 문자를 포함하고 있으므로 개행 문자를 제거해준다.

공백단위로 쪼개져 있는 명령어, 옵션, |, <, > & 를 파악하기 위해 각각의 flag를 만들고, 반복문을 통해 각각의 위치를 파악한다.

만약 파이프가 없는 경우
- 한가지 명령만을 수행한다.
- 자식 프로세스를 하나 생성한다.
- 자식 프로세스 :
  - '>' 플래그가 켜져있는 경우,
    - 표준 출력을 주어진 파일로 바꾸어 준다.
    - 주어진 명령어를 실행한다.
  - '<' 플래그가 켜져있는 경우,
    - 표준 입력을 주어진 파일로 바꾸어 준다.
    - 주어진 명령어를 실행한다.
  - 둘다 아닌 경우,
    - 주어진 명령어를 실행한다.
- 부모 프로세스 :
  - '&' 플래그가 켜져있는 경우,
    - 백그라운드 실행을 위해 waitpid 에 WNOHANG 옵션을 준다.
  - '&' 플래그가 꺼져있는 경우,
    - waitpid의 옵션에 0을 주어 wait과 같은 기능을 하도록 한다.

파이프가 있는 경우,
- 2개의 명령을 수행해야한다.
- 'l' 이후의 문자열을 2번째 명령으로 간주하고 새로운 캐릭터 포인터 배열로 만든다.
- 자식 프로세스를 생성한다.
- 자식 프로세스 :
  - 자손 프로세스를 생성한다.
  - 자손 프로세스 :
    - 파이프를 만든다.
    - 표준 출력을 파이프로 지정한다.
    - 처음 캐릭터 포인터 배열에 대해 "파이프가 없는 경우" 를 실행한다.
  - 자식 프로세스 :
    - 표준 입력을 파이프로 지정한다.
    - 자손 프로세스를 기다린다.
    - 새로만든 캐릭터 포인터 배열에 대해 "파이프가 없는 경우"를 실행한다.
  - (단, 자손과 자식 프로세스는 순차적으로 일어나야 하기 때문에, wait을 사용해 준다.)
- 부모 프로세스 :
  - '&' 플래그가 켜져있는 경우,
    - 백그라운드 실행을 위해 waitpid 에 WNOHANG 옵션을 준다.
  - '&' 플래그가 꺼져있는 경우,
    - waitpid의 옵션에 0을 주어 wait과 같은 기능을 하도록 한다.

컴파일 과정

0. 컴파일 확인

```
dongjin@gangdongjin-ui-MacBookAir minishell % gcc body.c -o body
dongjin@gangdongjin-ui-MacBookAir minishell % ./body
osh>
```

1. ls -l

```
dongjin@gangdongjin-ui-MacBookAir minishell % ./body
osh> ls -l
total 256
-rwxr-xr-x  1 dongjin  staff  50264  3 28 20:28 body
-rw-r--r--@ 1 dongjin  staff   8502  3 28 20:53 body.c
-rw-r--r--  1 dongjin  staff   1369  3 26 22:09 t_body.c
-rw-------  1 dongjin  staff    387  3 28 19:21 text
-rwxr-xr-x  1 dongjin  staff  49576  3 27 04:28 try
-rw-r--r--  1 dongjin  staff    626  3 28 18:49 try.c
osh>
```

2. ls -l &

"osh>" 가 출력된 이후에 결과가 출력된 것으로 보아 명령이 백그라운드로 실행되었음을 알 수 있다.

```
osh> ls -l &
[1] 58446
osh> total 256
-rwxr-xr-x  1 dongjin  staff  50264  3 28 20:28 body
-rw-r--r--@ 1 dongjin  staff   8502  3 28 20:53 body.c
-rw-r--r--  1 dongjin  staff   1369  3 26 22:09 t_body.c
-rw-------  1 dongjin  staff    387  3 28 19:21 text
-rwxr-xr-x  1 dongjin  staff  49576  3 27 04:28 try
-rw-r--r--  1 dongjin  staff    626  3 28 18:49 try.c
```

3. ls -l > text

```
osh> ls -l > text
osh> cat text
total 248
-rwxr-xr-x  1 dongjin  staff  50264  3 28 20:28 body
-rw-r--r--@ 1 dongjin  staff   8502  3 28 20:53 body.c
-rw-r--r--  1 dongjin  staff   1369  3 26 22:09 t_body.c
-rw-------  1 dongjin  staff      0  3 28 21:44 text
-rwxr-xr-x  1 dongjin  staff  49576  3 27 04:28 try
-rw-r--r--  1 dongjin  staff    626  3 28 18:49 try.c
osh>
```

4. ls > text &

3번 명령과 차별점을 주기 하기 위하여 ls 를 사용하였다.

```
osh> ls > text &
[1] 58455
osh> cat text
body
body.c
t_body.c
text
try
try.c
osh>
```

5. cat < text

4번 명령의 결과 text에 ls의 결과가 입력되었고, cat의 결과로 ls의 결과가 성공적으로 나오는 것을 볼 수 있다.

```
osh> cat < text
body
body.c
t_body.c
text
try
try.c
osh>
```

6. cat < text &

```
osh> cat < text &
[1] 58469
osh> body
body.c
t_body.c
text
try
try.c
```

7. ls -l | grep body

1번 명령의 결과 중, body가 들어가는 결과만을 출력한 것을 볼 수 있다.

```
osh> ls -l | grep body
-rwxr-xr-x  1 dongjin  staff  50264  3 28 20:28 body
-rw-r--r--@ 1 dongjin  staff   8502  3 28 20:53 body.c
-rw-r--r--  1 dongjin  staff   1369  3 26 22:09 t_body.c
osh>
```

8. ls -l | grep body &

```
osh> ls -l | grep body &
[1] 58481
osh> -rwxr-xr-x  1 dongjin  staff  50264  3 28 20:28 body
-rw-r--r--@ 1 dongjin  staff   8502  3 28 20:53 body.c
-rw-r--r--  1 dongjin  staff   1369  3 26 22:09 t_body.c
```

프로그램 소스 코드

(소스 파일 또한 메일에 첨부되어 있습니다.)

```c
/**
 * @file body.c
 * @author 2017012642 강동진
 * @brief
 *  [운영체제론] 1차 프로젝트 과제
 *  미니 쉘 구현하기.
 *
 *  -- 기능 --
 *  명령어 or 명령어 &
 *  명령어 > 파일 or 명령어 > 파일 &
 *  명령어 < 파일 or 명령어 < 파일 &
 *  명령어1 | 명령어2 or 명령어1 | 명령어2 &
 *  (단 명령어는 옵션을 포함함.)
 *
 * @version 0.1
 * @date 2021-03-28
 *
 * @copyright Copyright (c) 2021
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include <fcntl.h>

#define MAX_LINE 80 /* The maximum length command */

// 주어진 명령어를 실행하는 함수로, <, > 를 처리할 수 있다.
int exeInstruction(int RD_Lflag, int RD_Rflag, int BG_flag, char *argp[]);

int main(void)
{
    char *args[MAX_LINE / 2 + 1]; /* command line arguments */
    int should_run = 1;           /* flag to determine when to exit program */

    while (should_run)
    {
        printf("osh> ");
        fflush(stdout);

        // 명령어 파싱 (파이프 유무, & 유무, 명령어 분기 처리)
        char input[MAX_LINE];
        if (fgets(input, MAX_LINE, stdin) == NULL)
        {
            perror("fgets error");
        }

        // 공백 단위로 split
        char *strtok_p = strtok(input, " ");
        int args_length = 0;
        while (strtok_p != NULL)
        {
            args[args_length] = strtok_p;
            strtok_p = strtok(NULL, " ");
```

```
            args_length++;
        }

        // execvp 의 args의 마지막 인자로 NULL 이 필요하기 때문에 삽입해줌.
        args[args_length] = NULL;

        // 개행 문자 제거
        int leng = strlen(args[args_length - 1]);
        *(args[args_length - 1] + (leng - 1)) = '\0';

        // 명령어 탐색
        // pipe_flag : args에서 pipe의 위치
        int pipe_flag = 0;
        int BG_flag = 0;
        int RD_Lflag = 0;
        int RD_Rflag = 0;
        for (int i = 0; i < args_length; i++)
        {
            if (*args[i] == '|')
            {
                pipe_flag = i;
            }

            if (*args[i] == '&')
            {
                BG_flag = i;
            }

            if (*args[i] == '>' && RD_Rflag == 0)
            {
                RD_Rflag = i;
            }

            if (*args[i] == '<' && RD_Lflag == 0)
            {
                RD_Lflag = i;
            }
        }

        char *args2[MAX_LINE / 2 + 1];
        int RD_Lflag2 = 0;
        int RD_Rflag2 = 0;
        int BG_flag2 = 0;

        // pipe가 있는 경우, pipe이후의 명령을 처리하기 위한 전처리 과정
        if (pipe_flag)
        {

            args[pipe_flag] = NULL;

            for (int i = 1; i < (MAX_LINE / 2 + 1); i++)
            {
                if (args[pipe_flag + i] == NULL)
                {
                    args2[i - 1] = NULL;
                    break;
                }
                else
                {

                    args2[i - 1] = args[pipe_flag + i];
                }
            }

            for (int i = 0; args2[i] != NULL; i++)
```

```
            {
                if (*args2[i] == '>')
                {
                    RD_Rflag2 = i;
                }
                if (*args2[i] == '<')
                {
                    RD_Lflag2 = i;
                }
                if (*args2[i] == '&')
                {
                    BG_flag2 = i;
                }
            }
        }
        // & 제거
        if (BG_flag)
        {
            args[BG_flag] = NULL;
            args2[BG_flag2] = NULL;
        }
        // exit 검사
        if (strcmp(args[0], "exit") == 0)
        {
            // 같음
            should_run = 0;
            continue;
        }
        //
-------------------------------------------------------------------------------------------
        // pipe 없을 때
        if (pipe_flag == 0)
        {
            exeInstruction(RD_Lflag, RD_Rflag, BG_flag, args);
        }
        //
-------------------------------------------------------------------------------------------
        // pipe 있을 때
        else
        {
            pid_t child_pid = fork();
            if (child_pid < 0)
            {
                perror("pipe fork error");
            }
            else if (child_pid == 0)
            {
                // 1 : write
                // 0 : read
                int pipe_fd[2];
                if (pipe(pipe_fd) == -1)
                {
                    perror("pipe error");
                }
```

```
                pid_t child2_pid = fork();
                if (child2_pid < 0)
                {
                    perror("pipe fork error");
                }
                else if (child2_pid == 0)
                {
                    close(pipe_fd[0]);
                    dup2(pipe_fd[1], 1);
                    exeInstruction(RD_Lflag, RD_Rflag, 0, args);
                    return 0;
                }
                else
                {
                    close(pipe_fd[1]);
                    dup2(pipe_fd[0], 0);
                    exeInstruction(RD_Lflag2, RD_Rflag2, 0, args2);
                    return 0;
                }
```

```
                return 0;
            }
            else
            {
                // & 를 사용한 경우
                if (BG_flag)
                {
                    // not wait
                    int status;
                    int process_cnt = 1;
                    int retval = waitpid(-1, &status, WNOHANG);
                    if (retval == 0)
                    {
                        // 백그라운드로 실행됨을 알려주기 위함.
                        printf("[%d] %d\n", process_cnt, child_pid);
                    }
                }
                // & 를 사용하지 않는 경우
                else
                {
                    // wait
                    int status;
                    int retval = waitpid(-1, &status, 0);
                }
            }
        }

        // 반복
    }
    return 0;
}

//
==========================================================================================

int exeInstruction(int RD_Lflag, int RD_Rflag, int BG_flag, char *argp[])
{
    pid_t child_pid = fork();

    if (child_pid < 0)
    {
        perror("child_pid error");
        return -1;
    }
```

```
    else if (child_pid == 0)
    {
        // '<' 가 사용된 경우
        if (RD_Lflag)
        {
            int fd;
            // 파일이 없는 경우 에러
            fd = open(argp[RD_Lflag + 1], O_RDONLY | O_NONBLOCK, 0600);
            if (fd < 0)
            {
                perror("no file");
            }
            else
            {
                // 표준 입력을 파일로 바꿔준 다음, execvp가 처리할 수 있게 '<'를 null 로 바꿔
준다.
                dup2(fd, 0);
                argp[RD_Lflag] = NULL;
                execvp(argp[0], argp);
                perror("execvp Error Occured");
            }
            close(fd);
            return 0;
        }
        // '>' 를 사용한 경우.
        else if (RD_Rflag)
        {
            int fd;
            // 파일이 없는 경우 생성
            fd = open(argp[RD_Rflag + 1], O_CREAT | O_WRONLY | O_TRUNC, 0600);
            dup2(fd, 1);
```

```
            argp[RD_Rflag] = NULL;
            execvp(argp[0], argp);
            perror("execvp Error Occured");
            close(fd);
            return 0;
        }
        else
        {
            // '<' 와 '>' 가 동시에 사용된 경우.
            if (RD_Lflag && RD_Rflag)
            {
                perror("too many redirections ");
            }
            // 명령어만 실행하는 경우
            else
            {
                execvp(argp[0], argp);
                return 0;
            }
        }
    }
    else
    {
        // '&' 를 사용한 경우, WNOHANG 을 사용하여 background process 구현
        if (BG_flag)
        {
            // not wait
            int status;
            int process_cnt = 1;
            int retval = waitpid(child_pid, &status, WNOHANG);
            if (retval == 0)
            {
                printf("[%d] %d\n", process_cnt, child_pid);
```

```
            }
        }
        // '&' 를 사용하지 않은 경우
        else
        {
            // wait
            int status;
            int retval = waitpid(child_pid, &status, 0);
        }
    }
    return 0;
}
```