REPORT

2016311821 한승하

이번 Mini Shell 제작 Program Assignment Report입니다.

<Headers, Function prototypes, Global Variables>

```c
/* $begin shellmain */
#define MAXARGS   128
#define MAXLINE   256
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

/* function prototypes */
void eval(char *cmdline);
int parseline(char *buf, char **argv);
int builtin_command(char **argv);
void swsh_head(char **argv);
void swsh_tail(char **argv);
void swsh_cat(char **argv);
void swsh_cp(char **argv);
void swsh_mv(char **argv);
void swsh_rm(char **argv);
void swsh_cd(char **argv);
void swsh_pwd();
void swsh_exit(char **argv);
void close_pipe();

/* global variables */
int pipes[10][2];
pid_t pgid = 0;
```

다음은 이번 과제에서 사용한 header file, function prototypes, global variables입니다. Pipes는 pipe을 위해, pgid는 Group process id를 가져오기 위해 선언해 주었습니다.

<main>

```
int main()
{
    signal(SIGINT, SIG_IGN);
    signal(SIGTSTP, SIG_IGN);
    char cmdline[MAXLINE]; /* Command line */
        char* ret;
    while (1) {
        /* Read */
        printf("swsh> ");
        ret = fgets(cmdline, MAXLINE, stdin);
        if (feof(stdin) || ret == NULL)
            exit(0);

        /* Evaluate */
        eval(cmdline);
    }
}
```

Main 문에서는 signal 함수를 이용하여 무시할 signal 들 (SIGINT, SIGTSTP) 을 무시하도록 하여 주었습니다.

<Eval>

```
void eval(char *cmdline)
{
    char *argv[MAXARGS]; /* Argument list execve() */
        int pipe_argv[MAXARGS];
    char buf[MAXLINE];   /* Holds modified command line */
    int bg;              /* Should the job run in bg or fg? */
    int argc = 0;
        int idx,out_ptr = 0;
        int in_ptr =0;
        int is_in = 0;
        int is_out = 0;
        int pipe_wr = 0;
        int pipe_num = 0;
        int remain_pipe;
        int status;
        int is_child = 0;
    pid_t pid = 0;           /* Process id */
```

이번과제 가장 긴 Eval 함수의 시작 부분입니다. 함수내에서 사용할 값들을 선언해 주었습니다.

<Eval - Pipe>

```
    for(idx=0;idx<10;idx++)
    {
            if(pipe(pipes[idx]) <0){
                    perror("pipe error");
                    exit(-1);
            }
    }
strcpy(buf, cmdline);
bg = parseline(buf, argv);
if (argv[0] == NULL)
    return;   /* Ignore empty lines */
    else{}
    pipe_argv[pipe_wr++] = 0;
    while(argv[argc] != NULL)
    {
            if(strcmp("|",argv[argc]) == 0)
            {
                    pipe_argv[pipe_wr++] = argc+1;
            }
            argc++;
    }
    pipe_argv[pipe_wr] = argc+1;
    pipe_num = pipe_wr;
    remain_pipe = pipe_num;
```

Pipe를 처리하는 부분의 코드입니다. 먼저 전역 변수로 선언해준 pipes를 모두 pipe 함수를 이용해 pipe을 만들어 줍니다. 이후 "|"를 탐색해 pipe 해야 하는 개수와 각 command의 시작지점을 pipe_argv 에 저장해주었습니다 (pipe_argv는 int형 array로 argv에서 command 가 저장되어 있는 index를 저장할 수 있도록 하였습니다.)

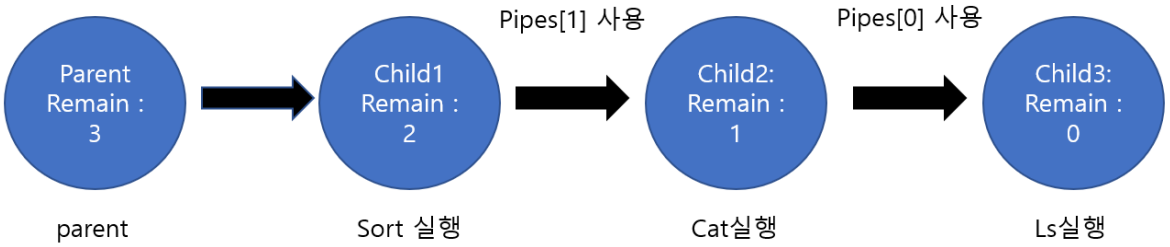
Pipe_num 은 실질적 실행을 해야 하는 command 개수를 저장하게 하였으며, pipe가 없을 경우 1 입니다.

```
if (!builtin_command(argv)) {
    if(pipe_num != 1)
    {
            while(remain_pipe != 0)
            {
                    pgid = getpgrp();
                    if ((pid = fork()) == 0)
                    {
                            is_child = 1;
                            setpgid(getpid(),pgid);
                            remain_pipe--;
                    }
                    else
                    {
                            if(is_child == 0) close_pipe(pipe);
                            break;
                    }
            }
            if(remain_pipe == 0)
            {
                    dup2(pipes[0][1],1);
                    close_pipe(pipe);
            }
            if(remain_pipe == pipe_num-1)
            {
                    dup2(pipes[pipe_num-2][0],0);
                    close_pipe(pipe);
            }
            for(idx=1;idx<pipe_num-1;idx++)
            {
                    if(idx == remain_pipe)
                    {
                            dup2(pipes[idx][1],1);
                            dup2(pipes[idx-1][0],0);
                            close_pipe(pipe);
                    }
            }
            if(pid != 0 && is_child)
            {
                    if (waitpid(pid, &status, WUNTRACED) < 0) printf("waitfg: waitpid error");
                    if(WIFSTOPPED(status)) kill(-1,SIGKILL);
            }
    }
```

Bulitin_command 가 아닐 경우에 함수 루틴을 시작합니다.

Pipe_num 이 1이 아닐 경우 즉 pipe가 존재할 경우 while 문을 돌며 실행해야 하는 command 만큼 fork를 할 수 있게 하였습니다. Remain_pipe는 초기에 pipe_num을 저장하고 있으며, fork을 할 때마다 child process 는 remain pipe 을 1식 줄이고 remain_pipe가 남았을 경우 child process 가 다시 fork하고, child의 child가 remain_pipe를 또 줄일 수 있게 하여, remain_pipe가 fork횟수를 규정해 줌과 동시에 else를 이용해 부모가 된 child가 remain_pipe 값을 유지할 수 있게 해주어 각 child process가 자신이 어떤 command를 맡고 있는지 구분해 줄 수 있게 하였습니다.

예를 들어 command 가 ls | cat | sort 일 경우 아래 그림과 같이 될 수 있도록 하였습니다.



따라서 각 fork를 한 child 또한 자신이 생성한 child를 기다리며 맨 앞 command 부터 순차적으로 실행하여 pipe를 이용해 stdout을 넘겨줄 수 있게 하였습니다.

```
void close_pipe()
{
        int idx;
        for(idx=0;idx<10;idx++)
        {
                close(pipes[idx][0]);
                close(pipes[idx][1]);
        }
}
```

ls_child 변수는 맨 상위 parent를 제외하고는 1이 될 수 있게 해주어, command 실행 및 pipe 이용 판단을 할 수 있게 해 주었습니다. 따라서 맨 상위 parent는 모든 pipe를 닫는 close_pipe() 함수를 호출합니다. Close_pipe의 구성은 위와 같습니다.

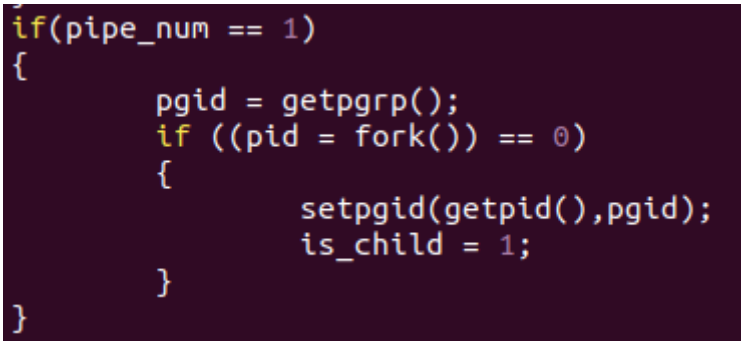
또한 매 fork시 getpgrp, getpid , setpgid를 이용하여 모든 child가 같은 process group에 속해 있도록 해 주었습니다.

이후 remain pipe의 값에 따라 dup2를 실행해줍니다. Pipe는 처음 실행되는 (맨 마지막 child) 부터 부모와의 통신에서 0번 / 1번 / 2번 순으로 쭉 사용하게 되는데, 여기서 특수하게 dup2를 해야 하는 child node는 맨 처음 실행되는 child (맨 마지막 child) 와 부모 바로 아래 달려있는 child 입니다. 맨 처음 child는 remain 값이 0으로, 이 child는 오직 stdout을 pipe로 넘겨주기만 합니다. 따라서 dup2를 stdout에 대해서만 실행합니다.

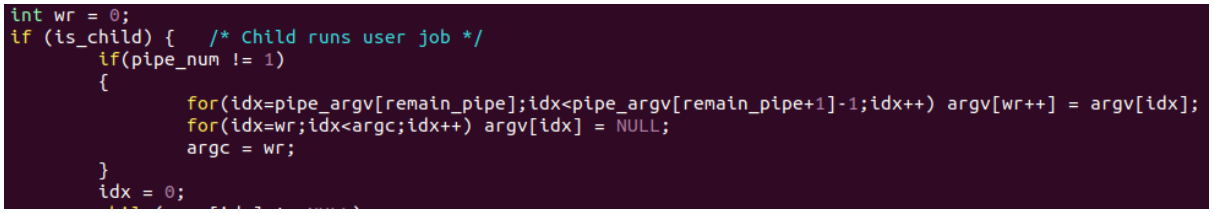
부모 바로 아래 child또한 특수하게 오직 stdin만을 바꿔주어야 합니다. 따라서 remain 이 1번만 감소되었을 경우 즉 pipe_num – 1 일 경우 stdin에 대해서만 dup2를 실행해 줍니다.

나머지 child node들은 자식과 연결된 pipe에 대해서 stdin, 부모와 연결된 pipe에 대해서 stdout을 dup2로 변경해 준 후, 모든 pipe를 close 하여, 파일이 끝났을 때 eof가 정상적으로 반환될 수 있게 하였습니다.

이후 맨 상위 parent node를 제외해고, (맨 상위 parent node는 built_in command가 아닐 경우 wait을 해주기 때문) wait으로 child가 먼저 종료되기를 기다리며 만일 ctrl – z 와 같은 이유로 죽었을 경우 kill(-1,sig)를 이용하여 같은 group의 모든 child가 죽게 하여 zombie process를 방지하여 주었습니다.



Pipe가 포함되지 않은 경우, fork는 1번만 진행되며, child에게는 is_child를 남겨 pipe가 있는 경우와 함께 범용적으로 처리해 줄 수 있게 하였습니다.



Pipe 가 존재하는 경우 remain_pipe 기준으로 실행해야 하는 command를 분리해 주었습니다. 이전에 저장해 놓은 pipe_argv의 값을 이용하여, argv[idx] 부터 다음 idx 전까지 (다음 idx는 "|" 다음 command의 index 이므로 다음 index-2까지) argv의 앞부분에 담아주고, 남은 부분을 NULL 처리하여 child 입장에서 마치 하나의 command만 포함된 argv가 들어온 것 처럼 보이게 하여 주었습니다.

<Eval – redirection>

```
while(argv[idx] != NULL)
{
        if(strcmp("<",argv[idx]) == 0)
        {
                is_in = 1;
                in_ptr = idx;
        }
        if(strcmp(">",argv[idx]) == 0)
        {
                is_out = 1;
                out_ptr = idx;
        }
        if(strcmp(">>",argv[idx]) == 0)
        {
                is_out = 2;
                out_ptr = idx;
        }
        idx++;
}
```

Redirection을 탐지하는 부분입니다. Redirection과 연관된 부호가 있을 경우 is_in / is_out 값과, ptr을 update해줍니다.

```
//redirection
if(argc>1){
        //input "<"
        if(is_in)
        {
                int redirection_input = open(argv[in_ptr+1],O_RDWR,0775);
                if(redirection_input == -1)
                {
                        perror("swsh");
                        exit(errno);
                }
                dup2(redirection_input,0);
                for(idx=in_ptr;idx<argc;idx++)
                {
                        argv[idx] = argv[idx+2];
                }
        }
        //output ">", ">>"
        if(is_out)
        {
                int redirection_output = open(argv[out_ptr+1],O_CREAT|O_RDWR,0775);
                if(is_out == 2) lseek(redirection_output,0,SEEK_END);
                dup2(redirection_output,1);
                for(idx=out_ptr;idx<argc;idx++)
                {
                        argv[idx] = argv[idx+2];
                }
        }
}
```

Redirection이 존재하는 경우 , 파일을 열어주고, dup2를 이용해 redirection을 적용해 준 다음,

argv에서 redirection기호와 인자를 삭제해 줍니다. >>인 경우 lseek을 이용하여 append될 수 있게 하여 주었습니다.

<Eval – Command>

```
//cmd_type1
if(strcmp("ls",argv[0]) == 0)   execvp(argv[0],argv);
else if(strcmp("man",argv[0]) == 0)     execvp(argv[0],argv);
else if(strcmp("grep",argv[0]) == 0) execvp(argv[0],argv);
else if(strcmp("sort",argv[0]) == 0) execvp(argv[0],argv);
else if(strcmp("awk",argv[0]) == 0)     execvp(argv[0],argv);
else if(strcmp("bc",argv[0]) == 0)      execvp(argv[0],argv);
else if(strncmp("./",argv[0],2) == 0)   execv(argv[0],argv);
//cmd_type2
else if(strcmp("head",argv[0]) == 0)    swsh_head(argv);
else if(strcmp("tail",argv[0]) == 0)    swsh_tail(argv);
else if(strcmp("cat",argv[0]) == 0)     swsh_cat(argv);
else if(strcmp("cp",argv[0]) == 0)      swsh_cp(argv);
//cmd_type3
else if(strcmp("mv",argv[0]) == 0)      swsh_mv(argv);
else if(strcmp("rm",argv[0]) == 0)      swsh_rm(argv);
//cmd_type4
else if(strcmp("pwd",argv[0]) == 0)     swsh_pwd();
else{
        fprintf(stderr,"%s: Command not found.\n", argv[0]);
        exit(0);
}
exit(0);
```

Cmd_type에 대해 독립적으로 실행될 수 있게 하여 주었습니다. Type1 그룹은 exec 계열 함수로 바꾸어 주었고, 나머지는 각각 swsh함수를 생성하여 처리해 주었습니다. exit함수는

```
int builtin_command(char **argv)
{
        if (!strcmp(argv[0], "quit")) exit(0);
        else if(strcmp("cd",argv[0]) == 0)
        {
                swsh_cd(argv);
                return 1;
        }
        else if(strcmp("exit",argv[0]) == 0)    swsh_exit(argv);
        else{}
        if (!strcmp(argv[0], "&"))    /* Ignore singleton & */
                return 1;
        return 0;                     /* Not a builtin command */
}
```

위와 같이 builtin에 포함되어 있습니다. 각 함수에 대해서는 추후에 설명하겠습니다.

```
    /* Parent waits for foreground job to terminate */
    if (!bg) {
        if (waitpid(pid, &status, WUNTRACED) < 0)
                    printf("waitfg: waitpid error");
    }
    else
        printf("%d %s", pid, cmdline);
}
if(WIFSTOPPED(status)) kill(-1,SIGKILL);
return;
```

맨 상위 parent는 child들이 끝나기를 기다리며, zombie가 생기지 않도록 위의 wait와 똑 같은 작업을 수행합니다.

<Command Function (swsh functions) >

<Head>

```
void swsh_head(char **argv)
{
        int is_stdin = 0;
        int remain_lines = 10;
        int fd_head = 0;
        char buf[1];
        if(argv[1] != NULL && strcmp(argv[1],"-n") == 0)
        {
                remain_lines = atoi(argv[2]);
                if(argv[3] == NULL) is_stdin = 1;
                else fd_head = open(argv[3], O_RDONLY, 0644);
        }
        else
        {
                if(argv[1] == NULL) is_stdin = 1;
                else fd_head = open(argv[1], O_RDONLY, 0644);
        }
        if(is_stdin)
        {
                while(read(0,buf,1))
                {
                        if(*buf == EOF) return;
                        if(*buf == '\n') remain_lines--;
                        if(remain_lines == 0)
                        {
                                if(write(1,"\n",1));
                                break;
                        }
                        if(write(1,buf,1));
                }
        }
        else{
                while(remain_lines>0)
                {
                        if(read(fd_head, buf, 1) <= 0) return;
                        if(*buf == '\n') remain_lines--;
                        if(write(1,buf,1));
                }
                close(fd_head);
        }
        return;
}
```

Head 함수는 우선 -n을 탐지하여, 있을 경우 remain_lines를 바꾸어 줍니다 (default 값은 10으로 선언해 주었습니다.) 이후 fd_head를 open 해 줍니다. Pipe를 위해 인자에 file이 없는 case를 만들어 주어 stdin을 받도록 하여 주었습니다.

Stdin일 때는 stdin에서, 아닐 때는 fd_head에서 1글자씩 읽으며 바로 stdout에 write해 줍니다. new_line이면 remain_lines를 감소시켜 줍니다. Remain_lines가 0이되면 종료해줍니다.

<Tail>

```
void swsh_tail(char **argv)
{
        int idx = 0;
        int remain_lines = 10;
        int total_word = 0;
        int *word_for_last_lines;
        int fd_head;
        int erase = 0;
        int is_stdin = 0;
        char buf[1];
        if(argv[1] != NULL && strcmp(argv[1],"-n") == 0)
        {
                word_for_last_lines = (int*)malloc(sizeof(int)*remain_lines);
                remain_lines = atoi(argv[2]);
                if(argv[3] == NULL) is_stdin = 1;
                else fd_head = open(argv[3], O_RDONLY, 0644);
        }
        else
        {
                word_for_last_lines = (int*)malloc(sizeof(int)*remain_lines);
                if(argv[1] == NULL) is_stdin = 1;
                else fd_head = open(argv[1], O_RDONLY, 0644);
        }
        if(is_stdin)
        {
                fd_head = open("temp.txt",O_CREAT|O_RDWR,0775);
                while(read(0,buf,1)) if(write(fd_head,buf,1));
                lseek(fd_head,0,SEEK_SET);
        }
        while(read(fd_head, buf, 1))
        {
                if(erase)
                {
                        erase = 0;
                        word_for_last_lines[idx] = 0;
                }
                word_for_last_lines[idx]++;
                if(*buf == '\n')
                {
                        idx = (idx+1)%remain_lines;
                        erase = 1;

                }
        }
        for(idx=0;idx<remain_lines;idx++) total_word = total_word - word_for_last_lines[idx];
        lseek(fd_head,total_word,SEEK_END);
        while(read(fd_head,buf,1)) if(write(1,buf,1));
        free(word_for_last_lines);
        close(fd_head);
        if(is_stdin)
        {
                if(fork() == 0) execl("/bin/rm","rm","temp.txt",NULL);
        }
        return;
}
```

Tail 함수입니다. Head와 마찬가지로 인자가 없을 경우 stdin을 받도록 하였으며, option을 체크하여 remain_lines를 설정해줍니다.

Tail은 각각 circler buffer에 10개를 계속 갱신해가며 저장하여, 종료되었을 시점에 맨 하위 10개 혹은 option이 있을 경우 remain_line개 만큼을 print해 줍니다. 이때 stdin 일 경우 temp.txt file을 만들어 read를 모두 기록해 넘겨주었고, word개수를 저장하여 lseek를 이용해 print해 줄 수 있게 하여 주었습니다. Temp.txt는 이후 rm을 이용해 삭제하여 주었습니다.

<cat>

```
void swsh_cat(char **argv)
{
        char buf[1];
        if(argv[1] == NULL)
        {
                while(1)
                {
                        if(read(0,buf,1) == 0) break;
                        if(*buf == EOF) return;
                        if(write(1,buf,1));
                }
        }
        else
        {
                int fd_head = open(argv[1], O_RDONLY, 0644);
                while(read(fd_head,buf,1)) if(write(1,buf,1));
                close(fd_head);
        }
}
```

cat함수는 stdin일 경우 read후 바로 write, eof일 경우 return 하게 해 주었습니다.

File일 경우 file을 끝까지 read/write 하게 하여 주었습니다.

<CP>

```
void swsh_cp(char **argv)
{
        char buf[1];
        int fd_src = open(argv[1], O_RDONLY, 0644);
        int fd_dest = open(argv[2], O_CREAT|O_WRONLY, 0664);
        while(read(fd_src,buf,1)) if(write(fd_dest,buf,1));
        close(fd_src);
        close(fd_dest);
}
```

Cp의 경우 src와 dest를 열어 src를 읽어 그대로 dest에 write하게 하여 주었습니다.

<MV>

```
void swsh_mv(char **argv)
{
        char* src = (char*)malloc(sizeof(char)*(strlen(argv[1])+3));
        char* dest = (char*)malloc(sizeof(char)*(strlen(argv[2])+3));
        strcpy(src,"./");
        strcpy(dest,"./");
        strcat(src,argv[1]);
        strcat(dest,argv[2]);
        strcat(src,"\0");
        strcat(dest,"\0");
        int result = rename(src,dest);
        free(src);
        free(dest);
        if(result != 0) perror("mv");

}
```

Mv의 경우 rename sys_call을 사용할 수 있도록 argv를 수정해 주었으며, rename을 사용해 이름을 바꾸어 주었습니다. 이때 perror로 error에 대해 error message를 stderr로 출력할 수 있게 해 주었습니다.

<CD>

```
void swsh_cd(char **argv)
{
        char* src = (char*)malloc(sizeof(char)*(strlen(argv[1])+3));
        strcpy(src,"./");
        strcat(src,argv[1]);
        strcat(src,"\0");
        int result = chdir(src);
        free(src);
        if(result != 0) perror("cd");
}
```

Cd의 경우 mv와 같이 argv를 수정하여 chdir을 사용해 주었습니다. 위와 마찬가지로 perror을 이용해 error message를 출력하여 주었습니다.

<PWD>

```
void swsh_pwd()
{
        char pwd_buf[1024];
        char* result = getcwd(pwd_buf,sizeof(pwd_buf));
        if(result == NULL) perror("pwd");
        else printf("%s\n",pwd_buf);
}
```

Pwd의 경우 getcwd를 pwdbuf에 저장하여, 출력하게 해 주었습니다. 이때 error인 경우 error message를 출력하게 해 주었습니다.

<Exit>

```
void swsh_exit(char **argv)
{
        int errorno = 0;
        if(argv[1] != NULL) errorno = atoi(argv[1]);
        if(write(2,"exit\n",5));
        exit(errorno);
}
```

Exit의 경우 errorno를 argv에서 받아와 exit print후 errorno로 exit하게 해 주었습니다.

감사합니다.

(보고서 이후 수정이 존재할 수도 있습니다)