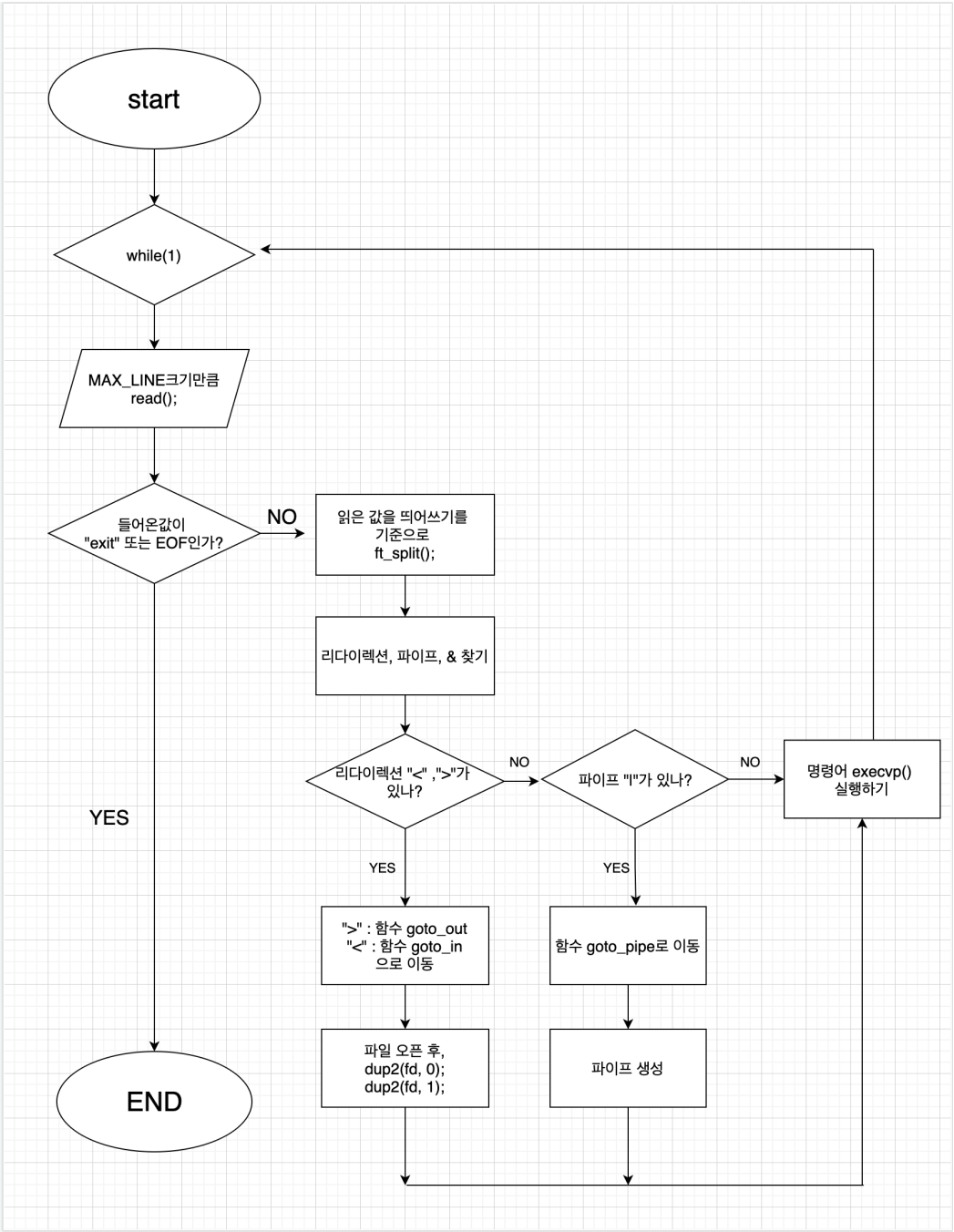
[Simple shell 알고리즘]

2017020628 김혜림



[프로그램 소스파일]

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

#define MAX_LINE 80

typedef struct  s_cmd //파이프나 리디렉션, & 가 있는지 확인하기위해 만들었습니다.
{
	int in;
	int out;
	int pipe;
	int background;
}			t_cmd;

void	init_cmd(t_cmd *cmd) // 초기화 시켜주기
{
	cmd->in = 0;
	cmd->out = 0;
	cmd->pipe = 0;
	cmd->background = 0;
}

size_t  count_word(char const *s, char c) // ft_split에 쓰이는 함수, 'c'를 기준으로 나눌때, 몇개의 칸이 필요한지 계산하는 함수
{
	size_t		count;

	count = 0;
	while (*s)
	{
		if (*s != c)
		{
			count++;
			while (*s && *s != c)
				s++;
		}
		else
			s++;
	}
	return (count);
}

void	put_word(char **result, char *start, size_t size, size_t i) // ft_split에서 쓰이는 함수, 'c'전까지의 문자열을 넣기위해 동적할당을 하고 strlcpy로 복사.
{
	if (!(result[i] = (char*)malloc(size)))
		return ;
	strlcpy(result[i], start, size);
}


char	**ft_split(char const *s, char c) // 문자열을 'c'를 기준으로 나누어주는 함수
{
```

```
	char	**result;
	char	*start;
	size_t	i;
	size_t	size;

	i = 0;
	if (s == 0)
		return (0);
	if (!(result = (char**)malloc(sizeof(char*) * count_word(s, c) + 1)))
		return (0);
	while (*s)
	{
		if (*s != c)
		{
			start = (char*)s; //현재 위치 저장하기
			while (*s && *s != c)
				++s;
			size = s - start + 1; //동적할당에 필요한 길이
			put_word(result, start, size, i++); // 'c' 전까지 문자열 넣기
		}
		else
			++s;
	}
	result[i] = 0; //마지막에 null 넣기
	return (result);
}


char	*make_null(char *s) // null문자 넣어주는 함수
{
	int i;

	i = 0;
	while (s[i])
	{
		if (s[i] == '\n')
			break;
		i++;
	}
	s[i] = '\0';
	return (s);
}

int	 find_cmd(char **s, t_cmd *cmd) // 리다이렉션과 파이프가 있는지 찾기
{
	int i = 0;

	while (s[i])
	{
		if (s[i][0] == '<')
		{
			cmd->in = 1;
			return i; //리다이렉션 위치 리턴
		}
		else if (s[i][0] == '>')
		{
```

```
                        cmd->out = 1;
                        return i;
                }
                else if (s[i][0] == '|')
                {
                        cmd->pipe = 1; //파이프의 위치 리턴
                        return i;
                }
                i++;
        }
        return 0;
}

void    goto_in(char **args, int pos) //리다이렉션 "<"
{
        int fd;

        fd = open(args[pos + 1], O_RDONLY); //파일열기
        if (fd < 0)
        {
                printf("fork failed\n");
                exit(1);
        }
        dup2(fd, 0); //stdin의 입력을 fd로 변경
        args[pos] = NULL; //명령어 뒤에 이상한 옵션이나 문자가 들어가지않도록 막기
        execvp(args[0], args); // 명령어 실행
        close(fd);
}

void    goto_out(char **args, int pos) //리다이렉션 ">"
{
        int fd; // 파일디스크립터

        fd = open(args[pos + 1], O_RDWR | O_CREAT | S_IROTH, 0644); //파일열기, 없다면 파일 생성하기
권한은 0644
        if (fd < 0)
        {
                printf("fork failed\n");
                exit(1);
        }
        dup2(fd, 1); //stdout의 출력을 fd로 보내기
        args[pos] = NULL; //명령어의 뒤에 이상한 옵션이나 문자가 들어가지않도록 막기
        execvp(args[0], args); // 명령어 실행
        close(fd);
}

void    goto_pipe(char **args, char **args2, int pos, t_cmd *cmd) // 파이프 처리하기
{
        int fd[2];
        pid_t pid2;
        int status;
        int i;

        pipe(fd);
        pid2 = fork();
        if (pid2 < 0)
```

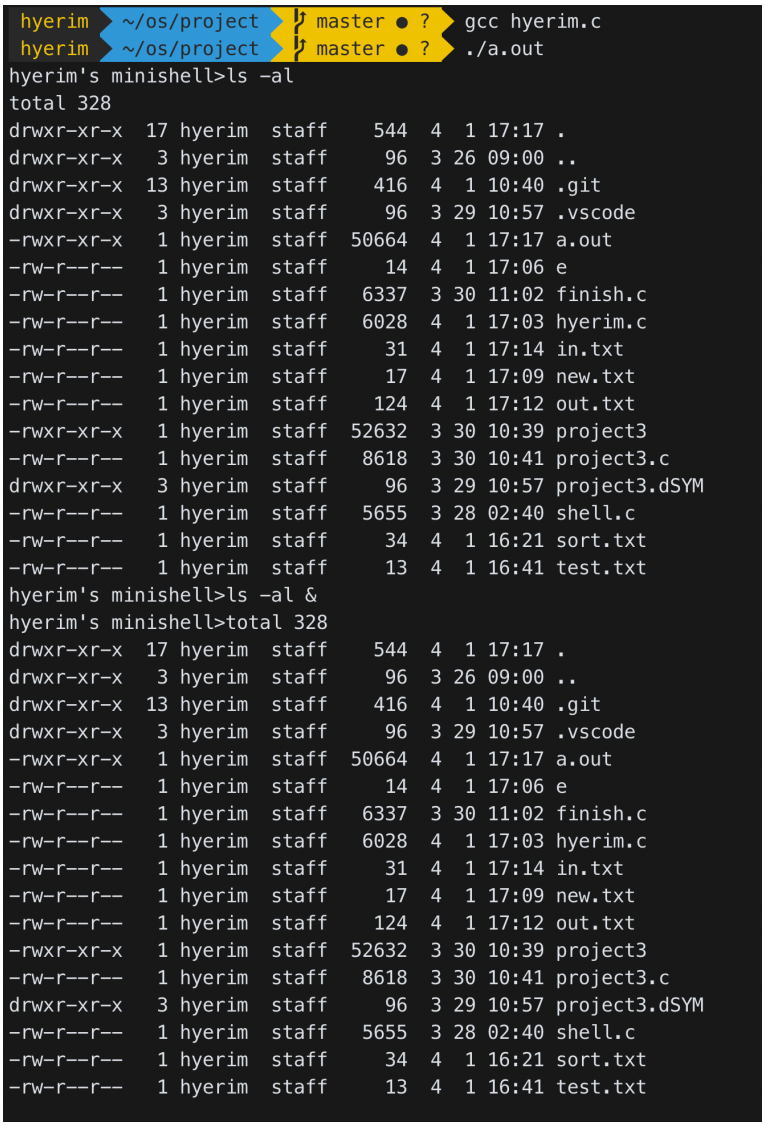
```
		{
			printf("fork faild\n");
			exit(1);
		}
		else if (pid2 == 0)
		{
			close(fd[0]); //안쓰는 fd[0] 닫아주기
			dup2(fd[1], 1); //stdout을 fd[1]로 바꾸기
			i = 0;
			while (args[i]) //첫번째 명령어의 뒤에 옵션 말고, 파이프나 그 뒤의 명령어가 들어가지않도
록 함
			{
				if (args[i][0] == '|')
					break;
				i++;
			}
			args[i] = NULL;
			execvp(args[0], args); //첫번쨔 명령어 실행
			close(fd[1]);
		}
		else if (pid2 > 0)
		{
			close(fd[1]); //안쓰는 fd[1]닫아주기
			dup2(fd[0], 0); //stdin을 fd[0]으로 바꾸기
			waitpid(pid2, &status, 0); //자식프로세스가 종료될때까지 기다리기
			execvp(args2[0], args2); //두번째 명령어 실행
			close(fd[0]);
		}
}

void	use_fork(char **args, int pos, t_cmd *cmd) // 자식프로세스 생성
{
		pid_t pid;
		int status;

		pid = fork();
		if (pid < 0)
		{
			printf("fork failed\n");
			exit(1);
		}
		else if (pid == 0) //자식프로세스
		{
			if (cmd->in == 1) // 리다이렉션 "<"이 있을경우
				goto_in(args, pos);
			else if (cmd->out == 1) //리다이렉션 ">"이 있을경우
				goto_out(args, pos);
			else if (cmd->pipe == 1) //}파이프 "|"가 있을경우
				goto_pipe(args ,args + pos + 1, pos, cmd);
			else
				execvp(args[0], args); // 그냥 명령어 하나있을 경우
			exit(0);
		}
		else if (pid > 0) //부모 프로세스
		{
			if (cmd->background == 1) // &명령어가 있을경우 자식 프로세스 기다리지 않기.
```

```
                        return ;
                waitpid(pid,&status,0);
        }
}

char    **rm_msg(char **args, t_cmd *cmd) // &명령어 찾고, &명령어를 NULL로 바꾸기.
{
        int i;

        i = 0;
        while (args[i])
        {
                if (!strcmp(args[i], "&"))
                {
                        cmd->background = 1;
                        args[i] = NULL;
                }
                i++;
        }
        return args;
}

int     main(void)
{
        char buf[MAX_LINE];
        char **args;
        char *new;
        t_cmd cmd;
        int pos;
        int i = 0;

        while (1)
        {
                init_cmd(&cmd);
                printf("hyerim's minishell>");
                fflush(stdout);
                if (read(0, buf, MAX_LINE) <= 0) //80byte씩 읽어옴
                        return 0;
                new = make_null(buf); //buf의마지막에 NULL 처리
                if (!strcmp(new, "exit")) //exit가 들어올경우 종료
                        exit(1);
                args = ft_split(new, ' '); //띄어쓰기를 기준으로 단어 나누기
                pos = find_cmd(args, &cmd); // 리다이렉션의 위치와 파이프 위치 찾기.
                args = rm_msg(args, &cmd); // &명령어 찾고, 지워주기.
                use_fork(args, pos, &cmd); //파이프나 리다이렉션이 있는경우.
        }
        return 0;
}
```

[컴파일과정과 명령어 실행]

&명령어가 들어오면, 부모프로세스가 자식프로세스를 기다리지않아,
사용자의 입력을 받을수있는 “hyerim’s minishell>” 이 출력된다.

1)컴파일, 명령어+옵션, 명령어+옵션 &



2) 명령어+옵션 > 파일명

cat new.txt로 내용을 출력해보면, hello world!가 개행없이 잘들어가있는 것을 확인할 수 있다.

```
hyerim's minishell>echo -n hello world! > new.txt
hyerim's minishell>cat new.txt
hello world!hyerim's minishell>
```

3) 명령어+옵션 > 파일명 &

cat new.txt로 확인해보면, !!!hello world!!!가 개행없이 잘 들어간것을 확인할 수 있다.

```
hyerim's minishell>echo -n !!!hello world!!! > new.txt &
hyerim's minishell>cat new.txt
!!!hello world!!!hyerim's minishell>
```

4) 명령어+옵션 < 파일명 과 명령어+옵션 < 파일명 &

sort의 “-r “ 옵션은 역순으로 정렬하여 출력한다.

```
hyerim's minishell>sort -r < sort.txt
9882
7374
6585
5495
200
123
1222

hyerim's minishell>sort -r < sort.txt &
hyerim's minishell>9882
7374
6585
5495
200
123
1222
```

5) 명령어+옵션 | 명령어+옵션 과 명령어+옵션 | 명령어+옵션 &

cat의 -b 옵션은 줄번호를 화면 왼쪽에 나타낸다. 실행결과는 sort -r 로 인해 줄번호의 역순으로 출력된다.

```
hyerim's minishell>cat -b test.txt | sort -r
     2  world!
     1  hello
hyerim's minishell>cat -b test.txt | sort -r &
hyerim's minishell>     2       world!
     1  hello
```

6) 프로젝트 예시에 나와있던 명령어들...

ls -l | less

```
total 328
-rwxr-xr-x  1 hyerim  staff  50664  4  1 17:17 a.out
-rw-r--r--  1 hyerim  staff     14  4  1 17:06 e
-rw-r--r--  1 hyerim  staff   6337  3 30 11:02 finish.c
-rw-r--r--  1 hyerim  staff   6028  4  1 17:03 hyerim.c
-rw-r--r--  1 hyerim  staff     31  4  1 17:14 in.txt
-rw-r--r--  1 hyerim  staff     17  4  1 17:09 new.txt
-rw-r--r--  1 hyerim  staff    124  4  1 17:28 out.txt
-rwxr-xr-x  1 hyerim  staff  52632  3 30 10:39 project3
-rw-r--r--  1 hyerim  staff   8618  3 30 10:41 project3.c
drwxr-xr-x  3 hyerim  staff     96  3 29 10:57 project3.dSYM
-rw-r--r--  1 hyerim  staff   5655  3 28 02:40 shell.c
-rw-r--r--  1 hyerim  staff     34  4  1 16:21 sort.txt
-rw-r--r--  1 hyerim  staff     13  4  1 16:41 test.txt
(END)
```

sort < in.txt

```
hyerim's minishell>cat in.txt
banana
apple
grape
melon
lemon
hyerim's minishell>sort < in.txt
apple
banana
grape
lemon
melon
hyerim's minishell>
```

ls > out.txt

```
hyerim's minishell>ls > out.txt
hyerim's minishell>cat out.txt
a.out
e
finish.c
hyerim.c
in.txt
new.txt
out.txt
project3
project3.c
project3.dSYM
shell.c
sort.txt
test.txt
t.txt
test.txt
hyerim's minishell>
```

감사합니다!!