운영체제 프로젝트 1 결과물 제출

ICT융합학부(컴퓨터 전공)

2017013072 윤지상

본 문서 전체 URL: https://github.com/beargrllys/3Grade1Semester/blob/master/OperatingSystem/%EC%9A%B4%EC%98%81%EC%B2%B4%EC%A0%9C%20%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8%201%20%EA%B2%B0%EA%B3%BC%EB%AC%BC%20%EC%A0%9C%EC%B6%9C.md

1.본인이 설계한 Simple Shell 알고리즘

이번 Simple Shell을 제작하기 위해 설계한 알고리즘은 다음과 같습니다.

---

0. 입력 과정

1. Scanf를 이용해 입력을 받는다.
2. 입력받은 문자열을 공백을 기준으로 나눈다.
3. 나누어진 문자열과 함께 자식프로세스를 실행한다.
4. 입력 문자열 맨 끝에 '&' 문자열이 있는지 확인하고 flag를 변경해준다.
5. 문자열 내에 특정 기호(|, <, >)를 검출하고 그에 맞는 함수를 실행해준다.
6. 한편 부모프로세스는 & flag에 맞추어 waitpid함수를 기다릴지 말지 결정한다.

1. osh> 명령어+옵션

1. 주어진 옵션들을 execlp에 맞게 문자열 배열로 가공한다.
2. 주어진 명령어를 execlp로 실행한다.
3. &가 붙어있을 경우, 부모 프로세스는 execlp가 종료될때까지 기다린다.

2. osh> 명령어+옵션 > 파일명

1. 맨 끝 문자열에 파일명을 이용해 File discriptor를 open한다.
2. 주어진 옵션들을 execlp에 맞게 문자열 배열로 가공한다.
3. Dup2(fd, STDOUT_FILENO) 함수를 이용해 출력값을 fd로 설정해준다.
4. 주어진 명령어를 execlp로 실행한다.
5. &가 붙어있을 경우, 부모 프로세스는 execlp가 종료될때까지 기다린다.

3. osh> 명령어+옵션 < 파일명 &

1. 맨 끝 문자열에 파일명을 이용해 File discriptor를 read한다.
2. 주어진 옵션들을 execlp에 맞게 문자열 배열로 가공한다.
3. 주어진 명령어과 읽어온 문자열을 이용해 execlp로 실행한다.
4. &가 붙어있을 경우, 부모 프로세스는 execlp가 종료될때까지 기다린다.

4. osh> 명령어+옵션 | 명령어+옵션

1. "|" 문자를 기준으로 양 문자열을 나누어 따로 parameter 배열을 만든다
2. 프로세스를 fork하고 자식과 부모간에 파이프를 생성한다.
3. 자식 프로세스가 앞쪽 명령어를 부모 프로세서가 뒷쪽 명령어를 처리하되 자식프로세서가 실행될 동안 부모는 자식이 끝날때까지 기다림으로서 자식이 파이프에 자신의 출력값을 모두 파이프에 주입하고 부모가 읽을 수 있도록한다.

4. 부모프로세스는 자식프로세스가 종료되고 파이프에 입력한 출력값을 읽어 와서 부모 본인의 명령어를 실행한다.
5. 주어진 명령어과 읽어온 문자열을 이용해 execlp로 실행한다.
6. &가 붙어있을 경우, main함수에서 fork된 부모 프로세스는 execlp가 종료될때까지 기다린다.

2. 프로그램 소스파일

이번 Simple Shell을 제작한 소스코드입니다.

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#define MAX_LINE 80
#define FPERM 0644
#define BUFF_SIZE 1024
#define READ_END 0
#define WRITE_END 1

/*--------------------------------
Department of Interdisciplinary Computing Technology
2017013072 Yun Ji Sang

Operating System 2021-1 / Project 1 Source Code
2021.03.28
---------------------------------*/

int std_input(char *inputBuf[]);
                    // Input and Tokenize command
int clear_args(char *inputBuf[]);
                     // reset all input Buffer & local Buffer
int print_input(char *inputBuf[], int *size);
                     // Print tokenized input data
void fileRedirection(char *inputBuf[], int *token, int *LtoR, int *argc);
                     // function for FileRedirection
void orderRedirection(char *inputBuf[], int *token, int *argc);
                     // function for File Content Redirection
int pipeline(char *inputBuf[], int *token, int *argc);
                    // fuction for Pipeline Operation
int seperateOrder(char *inputBuf[], char *front_order[], char *back_order[], int
*tocken, int *argc); // Seperate order (Command1 | Command2)
void normal_exec(char *inputBuf[]);
                     //Excute normal order
int whatCommand(int *argc, char *inputBuf[]);
                     //classified User order

int std_input(char *inputBuf[])
{
```

```
    int count = 0;
    char temp_str[MAX_LINE]; // save raw text line
    char *ptr;               // temporary save command piece

    clear_args(inputBuf);
    memset(temp_str, '\0', MAX_LINE);

    while (temp_str[0] == '\0')
    {
        printf("osh>");
        scanf("%[^\n]s", temp_str);
        getchar(); //absorb Enter buffer
    }
    ptr = strtok(temp_str, " ");
    while (ptr != NULL) // tokenize and save to input buffer
    {
        inputBuf[count] = (char *)malloc(sizeof(char) * MAX_LINE);
        strcpy(inputBuf[count], ptr);
        ptr = strtok(NULL, " ");
        count++;
    }
    return count - 1; // return command block size
}

int whatCommand(int *argc, char *inputBuf[])
{
    int count = 0;
    int token = 0; // The position of Special letter '|', '<>', exit
    int LtoR = 0;
    char *temp;
    while (*(inputBuf + count) != NULL) //Tour all command And execute
appropriate funtion
    {
        temp = *(inputBuf + count);
        if (strcmp(temp, ">") == 0) //file Redirection
        {
            LtoR = 1;
            fileRedirection(inputBuf, &token, &LtoR, argc);
        }
        else if (strcmp(temp, "<") == 0) //order Redirection
        {
            LtoR = 0;
            orderRedirection(inputBuf, &token, argc);
        }
        else if (strcmp(temp, "|") == 0) //piping mission
        {
            pipeline(inputBuf, &token, argc);
        }
        else if (strcmp(temp, "exit") == 0) //kill them all
        {
            kill(getppid(), SIGINT); // send signal to kill parent process
        }
        else
        {
            token++;
            count++;
        }
    }
```

```
    normal_exec(inputBuf); // If there is no special letter, exe by normal
command fuction
    return 0;
}

int clear_args(char *inputBuf[])
{
    int i = 0;
    while ((i < (MAX_LINE / 2 + 1)))
    {
        if (inputBuf[i] != NULL)
        {
            free(inputBuf[i]);  //free all allocated buffer from string buffer
pointer array
            inputBuf[i] = NULL; // and reset all allocated buffer address
        }
        i++;
    }
}

int print_input(char *inputBuf[], int *size)
{
    int count = 0;
    for (count = 0; count <= *size; count++)
    {
        printf("%s\n", *(inputBuf + count));
    }
}

void normal_exec(char *inputBuf[])
{

    if (execvp(inputBuf[0], inputBuf) == -1) // just execute normal command
    {
        perror("failed CHILD exec");
        exit(0);
    }
}

void fileRedirection(char *inputBuf[], int *token, int *LtoR, int *argc)
{
    int idx = 0;
    int fd, ret;
    char *paramList[MAX_LINE / 2 + 1];
    fd = open(*(inputBuf + *argc), O_RDWR | O_CREAT | O_TRUNC, S_IRUSR |
S_IWUSR); //read last File name for redirection
    if (fd < 0)
    {
        perror("file open error");
        exit(1);
    }

    for (idx = 1; idx < *token; idx++) // reset buffer before token
    {
        *(paramList + idx - 1) = *(inputBuf + idx);
    }
    for (int tmp = 0; tmp < idx + 1; tmp++) //copy all parameters buffer until
in front of token to local buffer,
```

```
    {
        paramList[tmp] = (char *)malloc(sizeof(char) * MAX_LINE);
        if (tmp == idx)
        {
            *(paramList + tmp) = NULL;
        }
        else if (tmp == idx + 1) // make end point last of paeameter list
        {
            *(paramList + tmp) = NULL;
        }
        else
        {
            strcpy(paramList[tmp], inputBuf[tmp]);
        }
    }
    if (dup2(fd, STDOUT_FILENO) == -1) // change target STDOUT to file
discriptor
    {
        perror("failed open");
        exit(1);
    }

    if (execvp(*paramList, paramList) == -1) // Excute all command
    {
        perror("failed exec");
        exit(1);
    }
    close(fd);
    ret = dup2(STDOUT_FILENO, fd);
    clear_args(paramList);
}

void orderRedirection(char *inputBuf[], int *token, int *argc)
{
    char *content = (char *)malloc(sizeof(char) * MAX_LINE);
    char *paramList[MAX_LINE / 2 + 1];
    char buf[BUFF_SIZE];
    int idx = *token;
    int fd;
    ssize_t rd_size;

    memset(content, '\0', MAX_LINE);

    if ((fd = open(*(inputBuf + *argc), O_RDONLY)) > 0) // read the file
contents
    {
        while (0 < (rd_size = read(fd, buf, BUFF_SIZE)))
        {
            strcat(content, buf);
            buf[rd_size] = 0;
        }
    }
    else
    {
        perror("file open error");
        exit(1);
    }
```

```
    close(fd);
    for (int tmp = 0; tmp < idx + 2; tmp++) //arrangement all parameter include
file readed content
    {
        paramList[tmp] = (char *)malloc(sizeof(char) * MAX_LINE);
        if (tmp == idx) // put the last parameter that read in file content
        {
            strcpy(paramList[tmp], buf);
        }

        else if (tmp == idx + 1)
        {
            *(paramList + tmp) = NULL;
        }
        else
        {
            strcpy(paramList[tmp], inputBuf[tmp]); //last buffer allocate from
input buffer
        }
    }
    if (execvp(inputBuf[0], paramList) == -1) //all arragement data executed
    {
        perror("failed exec");
        exit(1);
    }
    clear_args(paramList);
}

int pipeline(char *inputBuf[], int *token, int *argc)
{
    char *frontParam[MAX_LINE / 2 + 1];
    char *behindParam[MAX_LINE / 2 + 1];
    char buffer[BUFF_SIZE];
    int fd[2];
    int status = 2;
    int order2_size;
    pid_t pid;
    FILE *fp = NULL;
    size_t read_size = 0;

    order2_size = seperateOrder(inputBuf, frontParam, behindParam, token, argc);
// serperate two command between token "|"

    if (pipe(fd) == -1) //make the pipe
    {
        printf("PIPE ERROR\n");
        fprintf(stderr, "Pipe failed");
        return -1;
    }

    //READ_END 0
    //WRITE_END 1

    pid = fork(); //start child process

    if (pid < 0)
    {
        printf("FORK ERROR\n");
```

```
        fprintf(stderr, "Fork failed");
        return 1;
    }
    else if (pid == 0) // Child process Code
    {
        close(fd[READ_END]);                            //open WRITE_END
        if (dup2(fd[WRITE_END], STDOUT_FILENO) == -1) //change file discriptor
target to Pipe
        {
            perror("failed open");
            exit(1);
        }
        if (execvp(*frontParam, frontParam) == -1) //write & execute front
command output for parent process
        {
            perror("failed PARENT exec");
            exit(1);
        }
        close(fd[WRITE_END]); // writing Done
        //CHILD PROCESS
        exit(3);
    }
    else // Parent process Code
    {
        waitpid(-1, &status, 0); //waiting child process until front command
process Done
        printf(" ");
        close(fd[WRITE_END]);                          //open READ_END
        if (dup2(fd[READ_END], STDIN_FILENO) == -1) //push readed pipe data to
STDIN_FILENO
        {
            perror("failed open");
            exit(1);
        }
        close(fd[READ_END]); //close READ_END

        if (execvp(*behindParam, behindParam) == -1) //Excute behind command
output and output display
        {
            perror("failed CHILD exec");
            exit(0);
        }
    }
    clear_args(frontParam);
    clear_args(behindParam);
    return 0;
}

int seperateOrder(char *inputBuf[], char *front_order[], char *back_order[], int
*tocken, int *argc)
{ // serperate two command between token "|"
    int i = 0;
    int j = 0;
    for (i = 0; i < *tocken; i++)
    {
        front_order[i] = (char *)malloc(sizeof(char) * MAX_LINE);
        memset(front_order[i], '\0', MAX_LINE);
        strcpy(front_order[i], inputBuf[i]);
```

```
    }
    front_order[*tocken] = NULL;
    for (i = *tocken + 1; i <= *argc; i++)
    {
        back_order[j] = (char *)malloc(sizeof(char) * MAX_LINE);
        memset(back_order[j], '\0', MAX_LINE);
        strcpy(back_order[j], inputBuf[i]);
        //printf("%s  ==   %d   \n", back_order[j],  *tocken+1 + i);
        j++;
    }
    return *argc - *tocken + 1;
}

int main(void)
{
    char *args[MAX_LINE / 2 + 1]; /* command line arguments */
    int should_run = 1;
    int size, exp;
    int status;
    pid_t pid;

    for (int i = 0; i < (MAX_LINE / 2 + 1); i++)
    {
        args[i] = NULL;
    }

    while (should_run)
    {
        size = std_input(args);
        //print_input(args, &size);
        pid = fork(); // Execute by child process

        if (pid == 0)
        {
            pid = getpid();
            if (strcmp(args[size], "&") != 0) //distinguish the last letter is
"&"
            {
                whatCommand(&size, args);
            }
            else
            {
                printf("\nChild [%d]\n", pid); //notice background process is
started
                args[size] = NULL;              // excute without last letter "&"
                exp = size - 1;
                whatCommand(&exp, args);
            }
            return 0;
        }
        else if (pid > 0)
        {
            if (strcmp(args[size], "&") != 0)
            {
                waitpid(pid, &status, 0); // parent process waiting for child
process
            }
            else
```

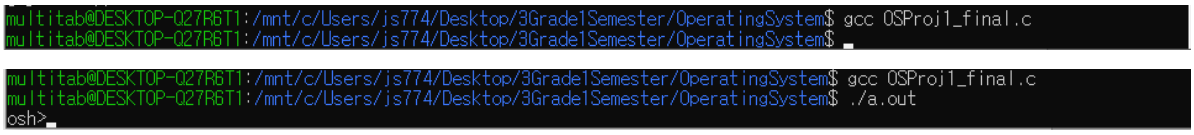
```
            {
                waitpid(-1, &status, WNOHANG); // parent process dose not
waiting for child process
            }
        }
        else
        {
            perror("FORK ERROR :");
            exit(0);
        }
        fflush(stdout); // all Done
    }
    return 0;
}
```

소스코드 업로드 Github

> https://github.com/beargrllys/3Grade1Semester/blob/master/OperatingSystem/OSProj1_final.c

3. 컴파일 과정을 보여주는 화면 캡처

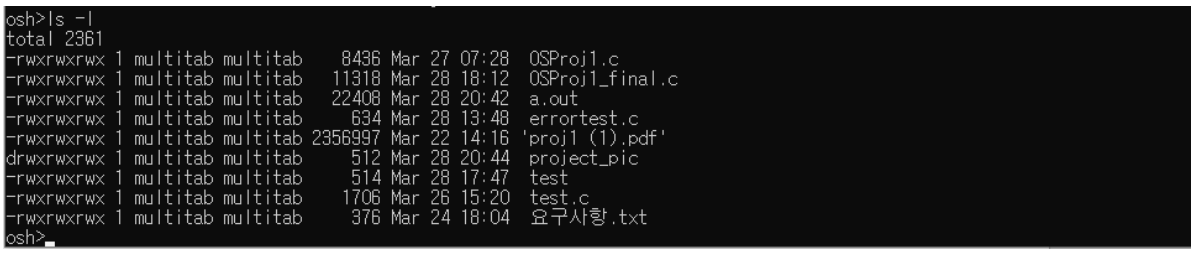
어떤 의미인지는 모르겠지만 컴파일 과정을 보여주는 화면입니다.



정상적으로 컴파일 되는것을 보실수 있습니다.

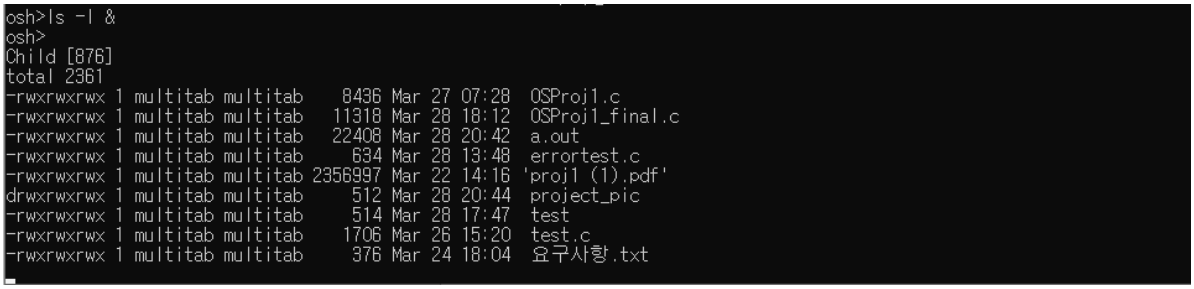
4. 주어진 형식을 한번씩 포함한 명령어들을 실행한 결과물과 그에 대한 간단한 설명

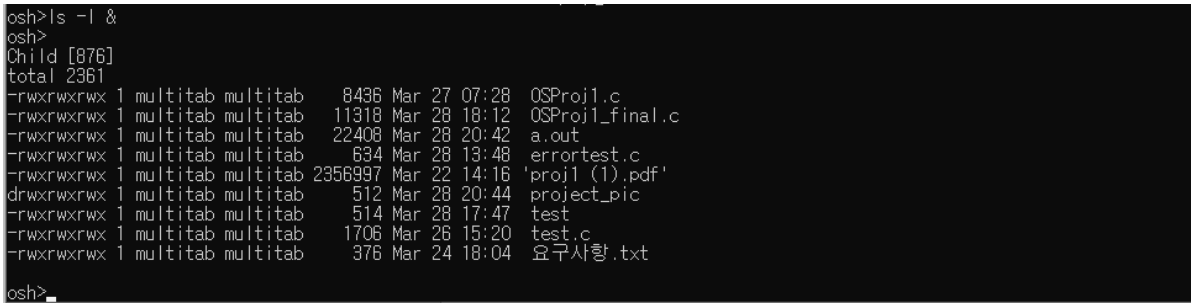
> 1. osh> 명령어+옵션



정상적으로 현재 디렉토리가 출력됩니다.

> 2. osh> 명령어+옵션 &

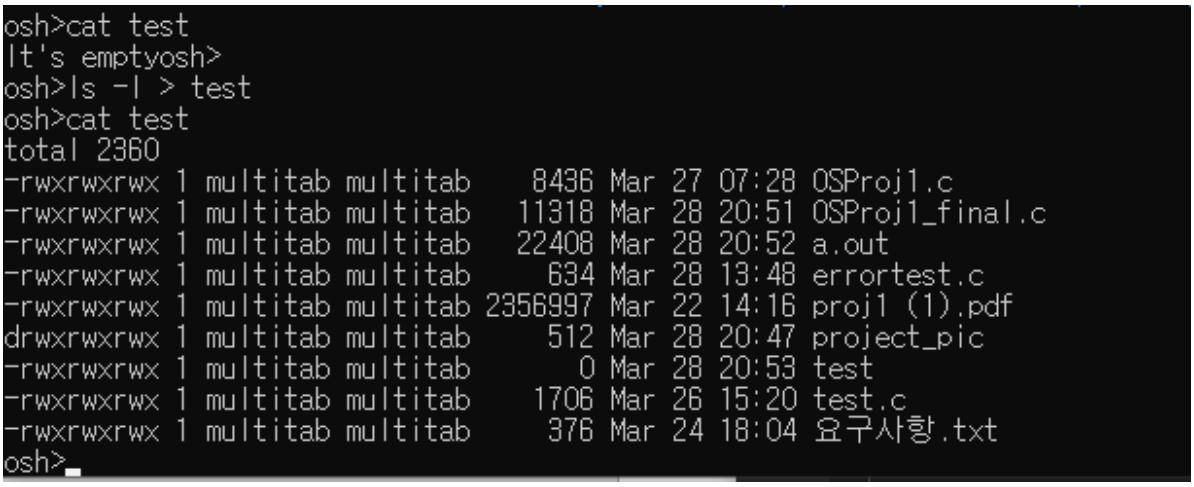




백그라운드로 실행되기에 "osh>"가 먼저 실행되고 후에 child process의 출력값이 출력됩니다.

"Child [876]"는 백그라운드 실행이 된다는 것을 의미하며 child process의 출력값이 나오고 한번 더 Enter를 입력하려 "osh>"가 나옵니다.

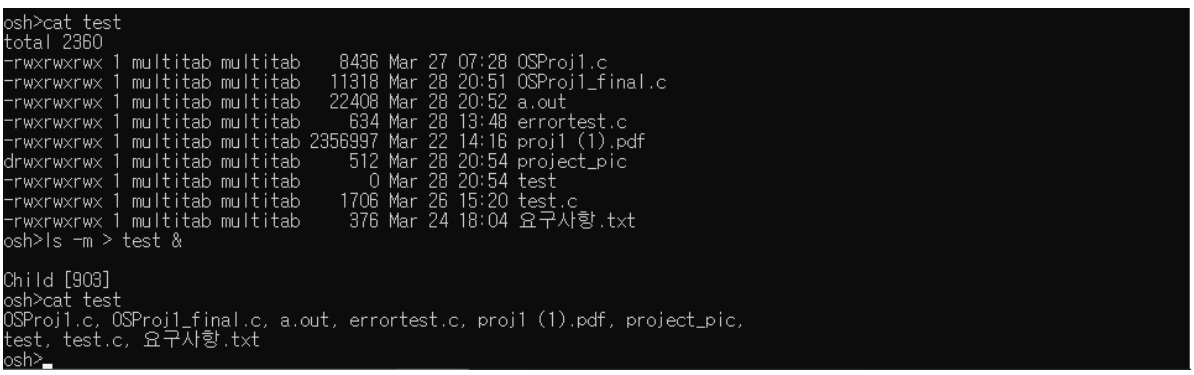
> 3. osh> 명령어+옵션 > 파일명



cat명령어를 통해 test파일의 내용을 확인하고 "ls -l >test" 명령어로 ls -l 의 출력을 test파일에 입력합니다.

명령어 실행 후에 test파일의 내용이 ls -l의 출력으로 변경된 것을 볼 수 있습니다.

> 4. osh> 명령어+옵션 > 파일명 &



백그라운드 프로세스로도 정상적으로 작동하는것을 확인 할 수 있습니다.

> 5. osh> 명령어+옵션 < 파일명

```
osh>cat test
/varosh>
osh>ls -l < test
total 0
drwxr-xr-x 1 root root   512 Apr 15  2020 backups
drwxr-xr-x 1 root root   512 Aug  5  2020 cache
drwxrwxrwt 1 root root   512 Aug  5  2020 crash
drwxr-xr-x 1 root root   512 Aug  5  2020 lib
drwxrwsr-x 1 root staff  512 Apr 15  2020 local
lrwxrwxrwx 1 root root     9 Aug  5  2020 lock -> /run/lock
drwxrwxr-x 1 root syslog 512 Aug  5  2020 log
drwxrwsr-x 1 root mail   512 Aug  5  2020 mail
drwxr-xr-x 1 root root   512 Aug  5  2020 opt
lrwxrwxrwx 1 root root     4 Aug  5  2020 run -> /run
drwxr-xr-x 1 root root   512 Jul 10  2020 snap
drwxr-xr-x 1 root root   512 Aug  5  2020 spool
drwxrwxrwt 1 root root   512 Aug  5  2020 tmp
osh>
```

test 파일의 내용을 "/var" 로 변경하고 "ls -l < test" 명령어를 실행합니다. /var 디렉토리의 파일들이 전시되는것을 볼 수 있습니다.

> 6. osh> 명령어+옵션 < 파일명 &

```
osh>cat test
/varosh>
osh>ls -l < test &

Child [909]
osh>total 0
drwxr-xr-x 1 root root   512 Apr 15  2020 backups
drwxr-xr-x 1 root root   512 Aug  5  2020 cache
drwxrwxrwt 1 root root   512 Aug  5  2020 crash
drwxr-xr-x 1 root root   512 Aug  5  2020 lib
drwxrwsr-x 1 root staff  512 Apr 15  2020 local
lrwxrwxrwx 1 root root     9 Aug  5  2020 lock -> /run/lock
drwxrwxr-x 1 root syslog 512 Aug  5  2020 log
drwxrwsr-x 1 root mail   512 Aug  5  2020 mail
drwxr-xr-x 1 root root   512 Aug  5  2020 opt
lrwxrwxrwx 1 root root     4 Aug  5  2020 run -> /run
drwxr-xr-x 1 root root   512 Jul 10  2020 snap
drwxr-xr-x 1 root root   512 Aug  5  2020 spool
drwxrwxrwt 1 root root   512 Aug  5  2020 tmp

osh>
```

백그라운드 프로세스로도 정상적으로 동작하는것을 확인 할 수 있습니다.

> 7. osh> 명령어+옵션 | 명령어+옵션

```
osh>ls -l | grep test
-rwxrwxrwx 1 multitab multitab     634 Mar 28 13:48 errortest.c
-rwxrwxrwx 1 multitab multitab       4 Mar 28 21:01 test
-rwxrwxrwx 1 multitab multitab    1706 Mar 26 15:20 test.c
osh>
```

ls -l의 출력 중 파이프를 통해 "test" 문자열이 들어가 있는것을 찾습니다. 정상적으로 작동하는것을 확인할수 있습니다.

> 8. osh> 명령어+옵션 | 명령어+옵션 &

```
osh>ls -l | grep out &

osh>Child [912]
-rwxrwxrwx 1 multitab multitab   22408 Mar 28 20:52 a.out

osh>
```

백그라운드 프로세스로도 정상적으로 동작하는것을 확인 할 수 있습니다.

전체 동작 검사 과정

```
multitab@DESKTOP-Q27R6T1:/mnt/c/Users/js774/Desktop/3Grade1Semester/OperatingSystem$ gcc OSProj1_final.c
multitab@DESKTOP-Q27R6T1:/mnt/c/Users/js774/Desktop/3Grade1Semester/OperatingSystem$ ./a.out
osh>ls -l
total 2360
-rwxrwxrwx 1 multitab multitab    8436 Mar 27 07:28  OSProj1.c
-rwxrwxrwx 1 multitab multitab   11318 Mar 28 20:51  OSProj1_final.c
-rwxrwxrwx 1 multitab multitab   22408 Mar 28 21:13  a.out
-rwxrwxrwx 1 multitab multitab     634 Mar 28 13:48  errortest.c
-rwxrwxrwx 1 multitab multitab 2356997 Mar 22 14:16 'proj1 (1).pdf'
drwxrwxrwx 1 multitab multitab     512 Mar 28 21:06  project_pic
-rwxrwxrwx 1 multitab multitab       4 Mar 28 21:01  test
-rwxrwxrwx 1 multitab multitab    1706 Mar 26 15:20  test.c
-rwxrwxrwx 1 multitab multitab     376 Mar 24 18:04  요구사항.txt
osh>ls -l &
osh>
Child [992]
total 2360
-rwxrwxrwx 1 multitab multitab    8436 Mar 27 07:28  OSProj1.c
-rwxrwxrwx 1 multitab multitab   11318 Mar 28 20:51  OSProj1_final.c
-rwxrwxrwx 1 multitab multitab   22408 Mar 28 21:13  a.out
-rwxrwxrwx 1 multitab multitab     634 Mar 28 13:48  errortest.c
-rwxrwxrwx 1 multitab multitab 2356997 Mar 22 14:16 'proj1 (1).pdf'
drwxrwxrwx 1 multitab multitab     512 Mar 28 21:06  project_pic
-rwxrwxrwx 1 multitab multitab       4 Mar 28 21:01  test
-rwxrwxrwx 1 multitab multitab    1706 Mar 26 15:20  test.c
-rwxrwxrwx 1 multitab multitab     376 Mar 24 18:04  요구사항.txt

osh>cat test
/varosh>
osh>ls -l > test
osh>cat test
total 2360
-rwxrwxrwx 1 multitab multitab    8436 Mar 27 07:28 OSProj1.c
-rwxrwxrwx 1 multitab multitab   11318 Mar 28 20:51 OSProj1_final.c
-rwxrwxrwx 1 multitab multitab   22408 Mar 28 21:13 a.out
-rwxrwxrwx 1 multitab multitab     634 Mar 28 13:48 errortest.c
-rwxrwxrwx 1 multitab multitab 2356997 Mar 22 14:16 proj1 (1).pdf
drwxrwxrwx 1 multitab multitab     512 Mar 28 21:06 project_pic
-rwxrwxrwx 1 multitab multitab       0 Mar 28 21:14 test
-rwxrwxrwx 1 multitab multitab    1706 Mar 26 15:20 test.c
-rwxrwxrwx 1 multitab multitab     376 Mar 24 18:04 요구사항.txt
osh>ls -l > test &

Child [996]
osh>ls -m > test &

Child [997]
osh>cat test
OSProj1.c, OSProj1_final.c, a.out, errortest.c, proj1 (1).pdf, project_pic,
test, test.c, 요구사항.txt
osh>cat dir
/home/multitabosh>
osh>ls -l < test
malloc(): corrupted top size
osh>ls -l < dir
total 1149
-rwxr-xr-x 1 multitab multitab    2421 Mar 28 02:13 04-getcwd.c
-rwxr-xr-x 1 multitab multitab    2421 Mar 28 02:12 Document
-rwxr-xr-x 1 multitab multitab    2421 Mar 28 02:12 Documents
-rwxr-xr-x 1 multitab multitab   17304 Mar 28 02:15 a.out
-rw------- 1 multitab multitab       0 Mar 24 10:35 bar
drwxr-xr-x 1 multitab multitab     512 Jan 31 00:05 bin
drwxr-xr-x 1 multitab multitab     512 Mar 28 02:13 dir
-rw-rw-rw- 1 multitab multitab       0 Mar 24 10:35 foo
-rwxr-xr-x 1 multitab multitab     102 Nov  3 11:55 go_window.sh
-rw-rw-r-- 1 testusr  multitab       0 Mar 23 13:40 test
-rwxrwxrwx 1 multitab multitab     514 Mar 24 10:33 umask.c
drwxr-xr-x 1 multitab multitab     512 Jun 29  2011 yum-3.4.3
-rw-r--r-- 1 multitab multitab 1140370 Dec  1  2018 yum-3.4.3.tar.gz
osh>ls -m < dir &

Child [1002]
osh>04-getcwd.c, Document, Documents, a.out, bar, bin, dir, foo, go_window.sh, test, umask.c, yum-3.4.3, yum-3.4.3.tar.g
z

osh>ls -l | grep out
-rwxrwxrwx 1 multitab multitab   22408 Mar 28 21:13 a.out
osh>ls -l | grep test &

osh>Child [1005]
-rwxrwxrwx 1 multitab multitab     634 Mar 28 13:48 errortest.c
-rwxrwxrwx 1 multitab multitab     107 Mar 28 21:15 test
-rwxrwxrwx 1 multitab multitab    1706 Mar 26 15:20 test.c

osh>
```