

리눅스 2주차 보고서

2025350021 스마트보안학부 박지우

프로그램 간단한 개요

- pridir() 로 현재 디렉토리 표시(입력할 때마다 표시 됩니다.)
- scan > addspace > exit , goback (백그라운드 문자 감지) > mulprom(다중 명령어 감지) > 실행 (반복)

Scan 함수

- 입력 받은후에 엔터키 제거
- 그리고 addspace 함수에서
- 다중명령어 | & 기호 양옆으로
스페이스바를 추가하여 파싱 할 때
스페이스바를 기준으로 파싱을
할 수 있게 함.
- 그리고 여기에서 백그라운드(&) 가
- 있으면 백그라운드로 보내버리고 입
- mulprom 으로 이동
- Exit 하면 종료

```
void scan(void) //스캔하고, 띄어쓰기 제거,  
토큰으로 분류  
{  
  
    char line[MAX];  
    char* nospace[MAX];  
  
    while (1)  
    {  
        if (!fgets(line, sizeof(line), stdin))  
            break;  
        line[strcspn(line, "\n")] = NULL; //  
        \n 제거  
        addspace(line);  
        int i = 0;  
        char* token = strtok(line, " ");  
        while (token != NULL)  
        {  
            nospace[i++] = token;  
            token = strtok(NULL, " ");  
        }  
        nospace[i] = NULL;  
        int bg = 0;  
        if (i > 0 && strcmp(nospace[i-1], "&") ==
```

goback

- 자식 프로세스 생성하고
- mulprom 으로 이동
- 가독성을 위해 만듦 scan 에서 처리해도 되긴 함

```
void goback(char* token[]) {  
    pid_t pid = fork();  
    if (pid < 0) {  
        perror("fork");  
        return;  
    }  
    if (pid == 0) {  
        // 자식 프로세스: 백그라운드로 멀티/  
        파이프라인 처리  
        mulprom(token);  
        exit(0);  
    }  
}
```

mulprom (1)

다중 명령어가 있는지 구분
하는 함수

&&||·을 인식해서 각각의
기호 뜻에 맞게 처리하도록
함.

이식된 다중 명령어 기준
양옆으로 큰 분리를
다중 명령어 오른쪽 새 토큰
으로 다시 mulprom으로 가서
다중 명령어 없을 때까지
재귀 호출 됨.

```
void mulprom(char* token[]) //다중 명령어 처리
= 다중 명령어 기준으로 왼쪽과 오른쪽으로 나눔
{
    int nomulti = 0;
    char* sorttok[MAX] = { 0 };
    char* sorttok2[MAX] = { 0 };
    for (int i = 0; token[i] != NULL; i++)
    {
        int endcheck = 0;
        if (strcmp(token[i], "&&") == 0)
        {
            ++nomulti;
            for (int a = 0; a < i; a++)
            {
                sorttok[a] = token[a];
            }

            for (int a = 0; token[a+i+1] != NULL; a++)
            {
                sorttok2[a] = token[a + i + 1];
                ++endcheck;
            }
            sorttok[i] = NULL; // 끝 체크
            sorttok2[endcheck] = NULL;
            int check;
            check = yesnpipe(sorttok);
            if (check == 0)
            {
                mulprom(sorttok2);
                break;
            }
            else break;
        }
        else if (strcmp(token[i], "||") == 0)
        {
            ++nomulti;
```

mulprom (2)

다중명령어 다 없앨 때까지 재귀호출 끝냈으면
이제 yesnpipe 로 이동

```
++nomulti;  
for (int a = 0; a < i; a++)  
{  
    sorttok[a] = token[a];  
}  
  
for (int a = 0; token[a + i + 1] !=  
NULL; a++)  
{  
    sorttok2[a] = token[a + i + 1];  
    ++endcheck;  
}  
sorttok[i] = NULL; // 끝 체크  
sorttok2[endcheck] = NULL;  
int check;  
check = yesnpipe(sorttok);  
  
mulprom(sorttok2);  
break;
```

yesnpipe

그냥 넘어온 토큰 개수(check1)이랑
파이프라을 뺀 토큰 개수(check2) 가 같은지
비교함 없으면 prompt 로 이동
있으면 dividpipe 로 이동

```
int yesnpipe(char* token[])
{
    int check1 = 0;
    int check2 = 0;
    for (int q = 0; token[q] != NULL; q++)
    {
        check1++;
    }
    for (int p = 0; token[p] != NULL; p++)
    {
        if (strcmp(token[p], "|") == 0) // |
            있으면 파이프로 ㄱㄱ
        {
            return dividpipe(token);
            break;
        }
        check2++;
    }
    if (check1 == check2) // check1==check2 즉
        파이프 없으면 그냥 없는걸로 ㄱㄱ
    {
        return prompt(token);
    }
}
```

dividepipe

- 파이프라인 개수 나누고
- 파이프라인에 관련된 인자들을
순서대로
- 이차원 배열에 나누어 담는 과정

```
int dividepipe(char* token[]) //파이프라인 나누기
{
    int a=0; // 파이프 개수
    for(int i=0;token[i]!=NULL;i++)
    {
        if(strcmp(token[i],"|")==0) //문자열을
            비교하기 위해 큰따옴표로 해야 한다고 함.
        {
            a++;
        }
    }
    int b = a+1; // b는 인자 개수
    if(b>20)
    {
        fprintf(stderr,"에러:파이프라인은 최대
            20개입니다.");
        return 0;
    }
    char* cmds[20][MAX]; // 각각의 인자들을 담는 배열
    int cmd=0,cmdline=0; // cmd는 명령어 단위, cmdline
    은 명령어 인자
    for (int i = 0; token[i] != NULL; i++) {
        if (strcmp(token[i], "|") == 0) {
            cmds[cmd][cmdline] = NULL; // 현재 명령어
            끝 표시
            cmd++;
            cmdline = 0;
        } else {
            cmds[cmd][cmdline++] = token[i];
        }
    }
    cmds[cmd][cmdline] = NULL;
```


pipeline

descrip 을 통해서 파이프 끝과 시작 지정
pipe() 가 채워줌
dup2 를 통해 파이프 데이터 갈아끼우고
close 로 다 쓴거 닫음
execvp 로 실제 명령을 실행함
wait(NULL) 을 통해 모든 자식 종료까지 대기
(좀비 프로세스를 어느정도 방지 가능!!!)

[WAIT\(\) : 네이버 블로그](#)

자식 프로세스 회수가 될 때까지 기다리려면
뭘 써야 할까 검색하다가 찾음.

```
void pipeline(char* cmds[][MAX], int cmd) {
    int pipes = cmd - 1;
    int descrip[pipes * 2];

    for (int i = 0; i < pipes; i++) {
        if (pipe(descrip + 2*i) < 0) {
            perror("pipe");
            return;
        }
    }

    for (int c = 0; c < cmd; c++) {
        pid_t pid = fork();
        if (pid < 0) {
            perror("fork");
            return;
        }
        if (pid == 0) {
            if (c > 0)
                dup2(descrip[2*(c-1)],
                    STDIN_FILENO);
            if (c < pipes)
                dup2(descrip[2*c + 1],
                    STDOUT_FILENO);
            for (int j = 0; j < 2*pipes; j++)
                close(descrip[j]);
            execvp(cmds[c][0], cmds[c]);
            perror("execvp");
            exit(1);
        }
    }
    for (int j = 0; j < 2*pipes; j++)
        close(descrip[j]);
    for (int c = 0; c < cmd; c++)
        wait(NULL);
}
```

prompt

Cd가 있으면 cd함수로 pwd 있으면 pwd 함수로 이동

만약에 둘 다 없으면 외부 함수로 인식하고 에러 뜨면 에러 출력

마찬가지로 자식 종료까지 대기(좀비 프로세스를 어느정도 대비 가능!!)

```
int num = 0; //token 인자 몇개인지
while (token[num] != NULL) //token 인자 개
세기
{
    num++;
}
for (int a = 0; token[a] != NULL; a++)
{
    if (num > 0 && strcmp(token[a], "cd")
    {
        return cd(token, num);
    }
    else if (num > 0 && strcmp(token[a],
    "pwd") == 0)
    {
        return pwd(token, num);
    }
    else { //외부 명령어 처리 없으면 오류
        pid_t pid = fork();
        if (pid < 0) {
            perror("fork");
            return 0;
        }
        if (pid == 0) {
            execvp(token[0], token);
            perror("execvp");
            exit(1);
        }
        wait(NULL);
        return 1;
    }
}
```

cd

prom 에서 cd가 달려 있으면 여기로 옴.
인자 1개면 홈 디렉토리, 2개면 2번째 인자의 디렉토리로 이동
3개면 2번째 인자가 -L, -P 가 아닐경우엔 오류 출력
하고
맞으면 3번째 인자의 디렉토리로 이동

```
{
    const char* target = NULL;
    int check;
    if (num > 3)
    {
        fprintf(stderr, "cd:too many 인자");
    }
    if (num == 1)
    {
        target = getenv("HOME");
        if (!target) //getenv 는 실패시 NULL 반환하니까
        {
            fprintf(stderr, "cd:이동 실패\n");
            return 0;
        }
    }
    else if (num == 2)
    {
        target = token[1];
    }
    else //인자가 3개인 경우 -> 예를들어 cd -L . 이런경우는 검색으로 알았습니다.
    {
        if (strcmp(token[1], "-L") == 0 || strcmp(token[1], "-P") == 0)
        {
            target = token[2];
        }
        else
        {
            fprintf(stderr, "cd: invalid option %s\n", token[1]);
        }
    }
}
```

pwd

그냥 현재 디렉토리 출력

```
int pwd(char* token[], int num)
{
    if (num > 1)
    {
        fprintf(stderr, "pwd error");
        return 0;
    }
    else
    {
        prindir();
    }
}
```

좀비 프로세스에 대한 고민

wait(NULL) 을 통해 외부 명령 한번 실행 -> 한번 회수를 할 수 있게 됨

어느정도의 좀비 프로세스를 막을 수 있을 거 같다고 생각함.