CyKor 과제 2주차

2025350212 임여준

개요

1. 함수 설명
2. 코드 설명
3. 코드 작성과정

함수 설명

1. Split 함수

```c
int split(char *input)
{

    int tcnt = 0;
    const char *delimiters[] = {"||", "&&", ";"};
    int delimiter_count = 3;

    const char *base = input;     // 원본
    const char *start = input;    // 미는거거

    while (*start) {
        const char *earliest = NULL;
        int earl_index = -1;

        // 모든 구분자에 대해 찾고, 가장 빨리 등장하는 구분자를 선택   -> 가장 먼저 나오는거 (나오면 저장하고 자르고 while문 진행방식식)
        for (int i = 0; i < delimiter_count; i++) {
            const char *pos = strstr(start, delimiters[i]);
            if (pos != NULL) {
                if (earliest == NULL || pos < earliest) {
                    earliest = pos;
                    earl_index = i;
                }
            }
        }

        if (earliest == NULL) {
            // 더 이상 구분자가 없으면 나머지 문자열을 토큰으로 저장
            if (*start) {
                tokens[tcnt] = strdup(start);
                tokens[tcnt][strcspn(tokens[tcnt], "\n")] = 0;
                tcnt++;
            }
            break;  //while 탈출
        } else {
            // 구분자 앞에 있는 문자열 (토큰)를 저장 (빈 문자열이 아닐 때)
            if (earliest > start) {
                tokens[tcnt] = strndup(start, earliest - start);
                tokens[tcnt][strcspn(tokens[tcnt], "\n")] = 0;
                tcnt++;
            }
            // 구분자 -> 토큰으로 저장
            tokens[tcnt] = strdup(delimiters[earl_index]);   // 구분자는 공백없이 저장됨됨
            tcnt++;

            // start를 구분자 바로 뒤로 이동   -> start 밀기
            start = earliest + strlen(delimiters[earl_index]);
        }
    }
    return (tcnt+1)/2;
}
```

'

여러 다중연산자가 한번줄에 여러 번 나왔을 때 해결 할 수 있기를 목표로 하고 코드를 짬.

가장 먼저 나오는 구분자를 기준으로 계속 반복해가며 token 이라는 전역변수

배열에 저장. -> token 배열에 다중연산자와 명령어들이 구분되어서 저장됨.
Return으로 명령어 개수를 출력함.

2. Separate 함수

```c
int seperate(const char *deli,char * input,char ** arg)
{
    int i =0;
    arg[i] = strtok(input, deli);
    while( arg[i] != NULL)
    {
        arg[++i] = strtok(NULL,deli);
    }
    return i;  // NULL 뺀 개수
}
```

입력받은 문자열을 입력받은 구분자를 기준으로 나눔.
Strtok 함수를 이용해 입력받은 delimiter 기준으로 arg 문자열 배열에 저장.

3. Recurs_pipe 함수

Pipe 를 구현하기 위해 재귀함수로 구성함.
Pipe 함수를 이용해 파일드라이버를 설정

이후 두개의 자식프로세스를 fork 함
왼쪽 자식 프로세스의 출력을 표준출력으로 리디렉션
오른쪽 자식 프로세스의 입력을 표준입력으로 리디렉션(둘다 dup2 함수이용)
왼쪽 자식은 처음 명령어를 실행 -> 출력 pipe 통해 오른쪽 자식의 입력으로 전단 -> 오른쪽 자식은 다음 명령어와 한 개 줄어든 카운터를 입력으로 하는 재귀함수 실행. -> 반복 -> 카운터 1인 경우 입력받은 명령어, 연결된 표준입력으로 멀티파이프라인 구현.

함수 제대로 실행됨을 확인하는 코드 -> 제대로 실행 -> 1return
재귀함수기에 함수 최종 실행시 원하는 리턴값 구현하는데 어려웠음.
부모 프로세스 -> wait함수 이용해 함수가 제대로 실행됐는지 확인함.

```c
int recurs_pipe(int cnt, char *commands[][10]) {
    if (cnt == 1) {
        execvp(commands[0][0], commands[0]);
        perror("execvp 실패 (마지막 명령어)");
        exit(EXIT_FAILURE);
    }

    int fd[2];
    if (pipe(fd) == -1) {
        perror("pipe 생성 실패");
        exit(EXIT_FAILURE);
    }

    pid_t left_pid = fork();
    if (left_pid < 0) {
        perror("fork 실패 (왼쪽)");
        exit(EXIT_FAILURE);
    }
    if (left_pid == 0) { // 왼쪽 자식 : 현재 명령어 실행, 출력은 파이프 쓰기 끝으로
        if (dup2(fd[1], STDOUT_FILENO) == -1) {
            perror("dup2 실패 (왼쪽, stdout)");
            exit(EXIT_FAILURE);
        }
        close(fd[0]);
        close(fd[1]);
        execvp(commands[0][0], commands[0]);
        perror("execvp 실패 (왼쪽)");
        exit(EXIT_FAILURE);
    }

    pid_t right_pid = fork();
    if (right_pid < 0) {
        perror("fork 실패 (오른쪽)");
        exit(EXIT_FAILURE);
    }
    if (right_pid == 0) { // 오른쪽 자식 : 파이프 읽기 끝을 표준 입력으로, 재귀 호출
        if (dup2(fd[0], STDIN_FILENO) == -1) {
            perror("dup2 실패 (오른쪽, stdin)");
            exit(EXIT_FAILURE);
        }
        close(fd[0]);
        close(fd[1]);
        recurs_pipe(cnt - 1, commands + 1);
        exit(EXIT_FAILURE);
    }

    // 부모 프로세스: 사용한 파이프의 fd를 닫고 자식 종료 대기
    close(fd[0]);
    close(fd[1]);

    int status_left, status_right;
    waitpid(left_pid, &status_left, 0);
    waitpid(right_pid, &status_right, 0);

    if (WIFEXITED(status_left) && WIFEXITED(status_right) &&
        WEXITSTATUS(status_left) == EXIT_SUCCESS &&
        WEXITSTATUS(status_right) == EXIT_SUCCESS) {
        return 1;
    }
    return 0;
}
```

4. Do_command 함수

```c
int do_command(char *input)   // return 1 실행행
{

    char dirc[MAX];


    char *pos1 = strchr(input, '&');  // | 로 나눈후
    int background = 0;
    if(pos1 != NULL)   // 백그라운드
    {
        int strNum;
        char * arg1[10];  // null 없음
        strNum = seperate("&",input,arg1);

        int status;
        background = 1;

        while (waitpid(-1, &status, WNOHANG) > 0)
            ;

        pid_t pid = fork();
        if (pid < 0) {
            perror("fork 실패");
            exit(EXIT_FAILURE);
        }

        if (pid == 0) {  // 자식 프로세스: 입력된 명령어 실행
            execlp("sh", "sh", "-c", arg1[0], (char *)NULL);
            perror("execlp 실패");
            exit(EXIT_FAILURE);
        } else {  // 부모 프로세스
            if (!background) {
                // 포그라운드 실행: 자식 프로세스가 종료될 때까지 대기
                waitpid(pid, &status, 0);
            } else {
                // 백그라운드 실행: 자식 프로세스의 종료를 기다리지 않음
                printf("백그라운드 작업 시작 (PID: %d)\n", pid);
            }
        }

        return 1;
    }

    char *pos = strchr(input, '|');  // | 로 나눈후
    if(pos != NULL)
    {
        int strNum;
        char * arg1[10];  // null 없음
        char * arg2[10][10];
        strNum = seperate("|",input,arg1);
        for(int i=0 ; i<strNum ; i++)
        {
            seperate(" ",arg1[i],arg2[i]); // " "로 또나눔 {"get","asd","NULL"} 느낌
        }

        return recurs_pipe(strNum,arg2);
    }

    char *arg[10];
    seperate(" ",input,arg);

    if(strcmp(arg[0], "exit")==0)
    {
        exit(0);
    }
```

```
        for(int i=0 ; i<strNum ; i++)
        {
            seperate(" ",arg1[i],arg2[i]); // " "로 또나눔 {"get","asd","NULL"} 느낌
        }

        return recurs_pipe(strNum,arg2);
    }

    char *arg[10];
    seperate(" ",input,arg);

    if(strcmp(arg[0], "exit")==0)
    {
        exit(0);
    }

    if(strcmp(arg[0], "cd")==0)
    {
        if(arg[1] == NULL || strcmp(arg[1], "~") == 0)
        {
            arg[1] = getenv("HOME");
        }

        if(chdir(arg[1]) == 0)
        {
            return 1;
        }
        else
        {
            perror("디렉토리 변경 실패 ");
            return 0;
        }
    }
    else if (strcmp(arg[0], "pwd")==0)
    {
        if (getcwd(dirc, sizeof(dirc)) != NULL)
        {
            printf("%s\n",dirc);
            return 1;
        }
        else
        {
            perror("fail");
            return 0;
        }
    }
    else
    {
        int status;
        pid_t pid = fork();
        if (pid == 0) {
            execvp(arg[0], arg);
            perror("명령 실행 실패");
            exit(EXIT_FAILURE);
        } else {
            wait(&status); // 부모     -> NULL 아니면 이상할때 존재. grep 같을때.
        }
        if (WIFEXITED(status)) return 1;   //  real 해결!   -> X
        // printf("%d catb\n",catBool);
        // if(catBool != 0) return 0;  // cat 없는파일 -> 해결     (자식프로세스 비정상 종료)    -> ls에서 오류류
        return 0;
    }
}
```

1. 백그라운드 작업 구현부분.
2. 파이프라인 -> | 확인 부분.
3. Exit 확인 부분
4. Cd, pwd 확인 부분
5. Exec 함수 구현.

➔ 입력값 sperate 함수로 나눔 -> do command 진행

5. Main 함수

```c
int main() {
    char dirc[MAX];
    while (1)
    {
        //getcwd(dirc,sizeof(dirc));
        printf("%s@%s:%s$ ",getenv("USER"),getenv("HOSTNAME"), getcwd(dirc, sizeof(dirc)));
        char input[MAX];

        fgets(input, MAX, stdin);
        input[strcspn(input, "\n")] = 0; // 입력 해결

        int input_num = split(input);

        // 다중명령어 조건실행
        int flag = 1;
        int result_bool;
        for (int i = 0; i < input_num; i++)
        {
            // printf("%d\n",flag);
            // i*2 내용 , i*2 +1 다중연산자
            if(flag)
            {
                result_bool = do_command(tokens[i*2]); // 그전에 flag 0 되면  변화X (; 아니면)   result bool 0,1 조건 세팅팅
                if ( i == input_num -1) break;
                if(strcmp(tokens[i*2+1], "||")==0)
                {
                    if(result_bool) flag = 0;   //세팅  return_bool 1이 참인거로 (실행된것)     cat asd -> return 값 1임..!!해결!!
                    else flag = 1;
                }
                else if(strcmp(tokens[i*2+1], "&&")==0)
                {
                    if(result_bool) flag = 1;   //세팅
                    else flag = 0;

                }
            }
            if ( i == input_num -1) break;
            if(strcmp(tokens[i*2+1], ";")==0) flag = 1;
        }
    }

}
```

일반적 리눅스 사용자 bash shell 구현 – printf(user,hostname,directory)

입력값 정리 -> 분할, 공백문자 제거

다중연산자 처리 -> for 문

Do_command 에서 명령어의 실현성공여부 리턴값으로 받음 -> ||,&& 구현

코드작성 과정

코드를 작성하며 가장 많이 고민하고 제가 원하는 것을 구현한 부분은 다중명령어 부분이였습니다. 이 파트(split 함수, main함수 for문, 여러함수의 리턴값(명령어 성공여부))관련 부분을 코딩하면서 다중명령어가 여러개일때, 다중명령어와 | 파이프라인이 같이 사용될때 모두 옳게 작동하는 코드를 작성하고 싶었습니다. 이에 다중 명령어 간의 관계와 우선순위, 진행 방식 등을 공부하였고 이를 코드에 구현해보았습니다.

특히 split함수를 코딩하면서 여러가지 구분자를 이용해 순서대로 구분자와, 명령어를 저장하기 위해 많은 고민을 했습니다. 이에 구조체, 2차원 배열 을 사용해서 구현도 해보며 많은 고민을 할 수 있었습니다.

Background 부분과 pipeline 부분을 코딩하면서 프로세스종료에 대한 고민을 많이 하였습니다. 프로세스 종료가 제대로 이루어지지 않아서 다음 출력값이 이상하게 나오는 경우도 많았습니다. 이를 해결하기 위해 여러 코드들을 추가하며 공부를 많이 하게 되었습니다.

코딩에서 아쉬운 부분 혹은 저의 실력 부족으로 구현하지 못한부분으로는 background 부분을 함수로 분리하고싶었지만 저의 실력부족으로 실현하지 못했습니다. 그 외에 cat 없는 파일 명령어 결과 실패를 리턴하고 싶었지만 이 또한 실력부족으로 실현하지 못했습니다.