Assignment3 : xsh(쉘 구현하기)

[AI기반시스템프로그래밍]

작성일 : 2025.10.30.

학번 : 23013393

작성자 : 최지혁

Assignment3 : xsh

> 이 과제는 **“Google Gemini : 2.5 Pro”**를 참고하여 작성하였습니다.

코드

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

// 최대 명령어 길이와 최대 인자 개수 설정
#define MAX_COMMAND_LENGTH 1024
#define MAX_ARGS 64

// ANSI 이스케이프 코드를 이용한 노란색 텍스트
#define ANSI_COLOR_YELLOW "\033[1;33m" // 굵게(1) + 노랑(33)
#define ANSI_COLOR_RESET "\033[0m"    // 모든 속성 초기화(0)
```

```
// 함수 선언
void print_prompt();
int parse_command(char *line, char **argv, int *is_background, char **infil
e, char **outfile);
int execute_builtin_command(char **argv);
void execute_external_command(char **argv, int is_background, char *infil
e, char *outfile);
void expand_environment_variables(char **argv);

/**
 * @brief 메인 REPL (Read-Eval-Print Loop) 함수
 * * 1. 프롬프트 출력 [cite: 68, 71]
 * 2. 사용자 입력 읽기 [cite: 69]
 * 3. 명령어 분석 및 실행 [cite: 69]
 * 4. EOF (Ctrl+D) 처리
 */
int main() {
    char line[MAX_COMMAND_LENGTH];
    int status;

    // REPL 루프 시작 [cite: 68]
    while (1) {
        // 백그라운드 프로세스 상태 점검 및 좀비 프로세스 방지
        // WNOHANG 옵션으로 블록되지 않고, 종료된 자식만 회수
        while (waitpid(-1, &status, WNOHANG) > 0) {
            // 자식 프로세스가 종료되었음을 알 수 있지만, 과제 요구사항에 명시된 출력
은 없으므로 생략
        }

        print_prompt(); // 프롬프트 출력 (PID + 디렉토리) [cite: 71]

        // 사용자 입력 읽기. Ctrl+D (EOF) 처리 [cite: 69, 70]
        if (fgets(line, MAX_COMMAND_LENGTH, stdin) == NULL) {
            // Ctrl+D (EOF) 입력 시 정상 종료
            printf("\n"); // 터미널 줄바꿈
            break;
        }
```

```
        // 입력된 명령어 라인 정리 (개행 문자 제거)
        line[strcspn(line, "\n")] = 0;
        if (strlen(line) == 0) continue; // 빈 줄 입력 시 재시작

        char *argv[MAX_ARGS];
        int is_background = 0;
        char *infile = NULL;
        char *outfile = NULL;

        // 명령어 라인 파싱 (인자, 리다이렉션, 백그라운드 여부 분리)
        int argc = parse_command(line, argv, &is_background, &infile, &outfil
e);
        if (argc == 0) continue;

        // 환경 변수 확장 (\$HOME, \$PATH 등)
        expand_environment_variables(argv);

        // 내장 명령어 실행 (exit, cd, pwd)
        if (execute_builtin_command(argv)) {
            continue; // 내장 명령어가 실행되었으면 외부 명령 실행 건너뛰기
        }

        // 외부 명령어 실행 (fork/exec/wait)
        execute_external_command(argv, is_background, infile, outfile);
    }

    return 0; // Ctrl+D 입력 시 정상 종료
}


/**
 * @brief 쉘 프롬프트 출력 함수
 * * 형식: xsh[PID]:현재디렉토리의마지막경로명> [cite: 13, 71]
 */
void print_prompt() {
    char cwd[1024];
    char *dir_name = "";
```

```
    // 현재 디렉토리 경로 가져오기
    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        // 경로 문자열에서 마지막 경로명만 추출. '/' 기준으로 찾기
        dir_name = strrchr(cwd, '/');
        if (dir_name != NULL) {
            dir_name++; // '/' 다음 문자부터 시작
        } else {
            dir_name = cwd; // 루트 디렉토리인 경우 전체 경로 사용
        }

        // HOME 디렉토리일 경우 '~'로 표시 (예시  참고)
        char *home_dir = getenv("HOME");
        if (home_dir != NULL && strcmp(cwd, home_dir) == 0) {
            dir_name = "~";
        }
    } else {
        // 에러 발생 시
        dir_name = "unknown";
    }

    // PID와 디렉토리 정보를 포함하여 프롬프트 출력
    printf("xsh[%d]:%s> ", getpid(), dir_name); // 현재 프로세스 PID 표시
    fflush(stdout); // 버퍼 비우기
}


/**
 * @brief 명령어 라인 파싱 함수
 * * 입력 라인을 인자 배열로 분리하고, 리다이렉션 (<, >) 및 백그라운드 (&) 기호를
파싱합니다.
 * @return 인자의 개수 (argc)
 */
int parse_command(char *line, char **argv, int *is_background, char **infil
e, char **outfile) {
    int argc = 0;
    char *token;
    char *saveptr;
```

```
    // 공백을 기준으로 토큰 분리
    token = strtok_r(line, " \t", &saveptr);
    while (token != NULL) {
        if (strcmp(token, "&") == 0) {
            // 백그라운드 실행 기호 (&)
            *is_background = 1;
        } else if (strcmp(token, "<") == 0) {
            // 입력 리다이렉션 (<)
            token = strtok_r(NULL, " \t", &saveptr);
            if (token) *infile = token;
        } else if (strcmp(token, ">") == 0) {
            // 출력 리다이렉션 (>)
            token = strtok_r(NULL, " \t", &saveptr);
            if (token) *outfile = token;
        } else {
            // 일반 명령어 인자
            argv[argc++] = token;
        }

        // 다음 토큰 가져오기
        token = strtok_r(NULL, " \t", &saveptr);
    }

    argv[argc] = NULL; // execvp를 위해 인자 리스트의 끝은 NULL로 설정
    return argc;
}

/**
 * @brief 환경 변수 (\$VAR) 확장 함수
 * * argv 배열을 순회하며 "$변수명" 패턴을 환경 변수 값으로 대체합니다.
 */
void expand_environment_variables(char **argv) {
    for (int i = 0; argv[i] != NULL; i++) {
        if (argv[i][0] == '$') {
            // '$' 다음 문자열(변수명)을 가져와 환경 변수 값을 찾음
            char *var_name = argv[i] + 1;
            char *value = getenv(var_name);
```

```
            if (value != NULL) {
                // 환경 변수 값이 존재하면 그 값으로 대체
                argv[i] = value;
            } else {
                // 환경 변수가 정의되지 않은 경우 빈 문자열로 처리하거나 그대로 둠
                // 과제 예시 [cite: 153]에 따라 공백으로 출력되는 것을 참고하여 그대로 둠
            }
        }
    }
}


/**
 * @brief 내장 명령어 실행 함수
 * * exit, cd, pwd와 같은 쉘 내장 명령어를 처리합니다.
 * @return 1 (내장 명령어를 실행했음), 0 (내장 명령어가 아님)
 */
int execute_builtin_command(char **argv) {
    if (strcmp(argv[0], "exit") == 0) {
        // 'exit' 명령어: 쉘 종료
        exit(0);
    } else if (strcmp(argv[0], "pwd") == 0) {
        // 'pwd' 명령어: 현재 작업 디렉토리 출력
        char cwd[1024];
        if (getcwd(cwd, sizeof(cwd)) != NULL) {
            printf("%s\n", cwd);
        } else {
            perror("pwd error");
        }
        return 1;
    } else if (strcmp(argv[0], "cd") == 0) {
        // 'cd' 명령어: 디렉토리 변경
        char *target_dir = argv[1];
        if (target_dir == NULL) {
            // 인자가 없으면 HOME 디렉토리로 이동
            target_dir = getenv("HOME");
        }
```

```
        if (target_dir != NULL && chdir(target_dir) != 0) {
            // 디렉토리 변경 실패 시 에러 출력
            perror("cd error");
        }
        return 1;
    }

    return 0; // 내장 명령어가 아님
}



/**
 * @brief 외부 명령어 실행 함수
 * * fork/exec/waitpid를 사용하여 외부 프로그램을 실행하고 리다이렉션을 처리합니다.
 */
void execute_external_command(char **argv, int is_background, char *infile, char *outfile) {
    pid_t pid;
    int status;

    // 자식 프로세스 생성
    if ((pid = fork()) == 0) {
        // **자식 프로세스**

        // 1. 리다이렉션 처리
        if (infile != NULL) {
            // 입력 리다이렉션 (<)
            int fd_in = open(infile, O_RDONLY);
            if (fd_in < 0) {
                fprintf(stderr, "xsh: No such file or directory: %s\n", infile);
                exit(1);
            }
            // 표준 입력 (STDIN_FILENO)을 파일 디스크립터로 대체
            dup2(fd_in, STDIN_FILENO);
            close(fd_in);
        }
```

```
        if (outfile != NULL) {
            // 출력 리다이렉션 (>)
            // 파일이 없으면 생성, 있으면 덮어쓰기 (O_TRUNC), 쓰기 전용 (O_WRONL
Y)
            int fd_out = open(outfile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
            if (fd_out < 0) {
                perror("xsh: failed to open output file");
                exit(1);
            }
            // 표준 출력 (STDOUT_FILENO)을 파일 디스크립터로 대체
            dup2(fd_out, STDOUT_FILENO);
            close(fd_out);
        }

        // 2. 외부 명령 실행
        // PATH 환경 변수에서 실행 파일을 찾음 (execvp)
        execvp(argv[0], argv);

        // execvp가 실패했을 경우 (명령어를 찾지 못했거나 실행 권한이 없는 경우)
        fprintf(stderr, "xsh: command not found: %s\n", argv[0]);
        exit(1); // 자식 프로세스 종료
    } else if (pid > 0) {
        // **부모 프로세스**

        if (is_background) {
            // 백그라운드 실행 (&)
            // 보너스: 시작 메시지를 노란색으로 출력
            printf(ANSI_COLOR_YELLOW "[bg] started pid=%d\n" ANSI_COLOR
_RESET, pid);
            // 쉘은 기다리지 않고 프롬프트로 즉시 복귀
        } else {
            // 포그라운드 실행
            // 자식 프로세스가 끝날 때까지 대기
            do {
                waitpid(pid, &status, 0); // 0 옵션으로 블록 (대기)
            } while (!WIFEXITED(status) && !WIFSIGNALED(status));
        }
    } else {
```

```
        // fork 실패 시
        perror("fork error");
    }
}
```

코드 구현

1. REPL루프 / EOF(Ctrl + D) 처리

```
int main() {
    char line[MAX_COMMAND_LENGTH];
    int status;

    // REPL 루프 시작 [cite: 68]
    while (1) {
        // 백그라운드 프로세스 상태 점검 및 좀비 프로세스 방지
        // WNOHANG 옵션으로 블록되지 않고, 종료된 자식만 회수
        while (waitpid(-1, &status, WNOHANG) > 0) {
            // 자식 프로세스가 종료되었음을 알 수 있지만, 과제 요구사항에 명시된 출력은 없으므로 생략
        }

        print_prompt(); // 프롬프트 출력 (PID + 디렉토리) [cite: 71]

        // 사용자 입력 읽기. Ctrl+D (EOF) 처리 [cite: 69, 70]
        if (fgets(line, MAX_COMMAND_LENGTH, stdin) == NULL) {
            // Ctrl+D (EOF) 입력 시 정상 종료
            printf("\n"); // 터미널 줄바꿈
            break;
        }

        // 입력된 명령어 라인 정리 (개행 문자 제거)
        line[strcspn(line, "\n")] = 0;
        if (strlen(line) == 0) continue; // 빈 줄 입력 시 재시작
```

```
        char *argv[MAX_ARGS];
        int is_background = 0;
        char *infile = NULL;
        char *outfile = NULL;

        // 명령어 라인 파싱 (인자, 리다이렉션, 백그라운드 여부 분리)
        int argc = parse_command(line, argv, &is_background, &infile, &outfil
e);
        if (argc == 0) continue;

        // 환경 변수 확장 (\$HOME, \$PATH 등)
        expand_environment_variables(argv);

        // 내장 명령어 실행 (exit, cd, pwd)
        if (execute_builtin_command(argv)) {
            continue; // 내장 명령어가 실행되었으면 외부 명령 실행 건너뛰기
        }

        // 외부 명령어 실행 (fork/exec/wait)
        execute_external_command(argv, is_background, infile, outfile);
    }

    return 0; // Ctrl+D 입력 시 정상 종료
}
```

- `main` 함수 내부에 `while(1)` 루프를 통해 **REPL**을 구현했습니다.
- `fgets`가 `NULL`을 반환하면 **EOF (Ctrl+D)** 입력으로 판단하고 `break`를 통해 쉘을 정상 종료합니다.
- 백그라운드 프로세스가 좀비 프로세스가 되는 것을 방지하기 위해, 루프 시작 시 `waitpid(-1, &status, WNOHANG)`를 호출하여 종료된 자식 프로세스를 비동기적으로 회수합니다.

2. 프롬프트 (PID + 디렉토리)

```
void print_prompt() {
    char cwd[1024];
```

```
    char *dir_name = "";

    // 현재 디렉토리 경로 가져오기
    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        // 경로 문자열에서 마지막 경로명만 추출. '/' 기준으로 찾기
        dir_name = strrchr(cwd, '/');
        if (dir_name != NULL) {
            dir_name++; // '/' 다음 문자부터 시작
        } else {
            dir_name = cwd; // 루트 디렉토리인 경우 전체 경로 사용
        }

        // HOME 디렉토리일 경우 '~'로 표시 (예시  참고)
        char *home_dir = getenv("HOME");
        if (home_dir != NULL && strcmp(cwd, home_dir) == 0) {
            dir_name = "~";
        }
    } else {
        // 에러 발생 시
        dir_name = "unknown";
    }

    // PID와 디렉토리 정보를 포함하여 프롬프트 출력
    printf("xsh[%d]:%s> ", getpid(), dir_name); // 현재 프로세스 PID 표시
    fflush(stdout); // 버퍼 비우기
}
```

- `print_prompt` 함수에서 프롬프트 출력을 담당합니다.
- `getpid()`를 사용하여 **현재 프로세스의 PID**를 표시합니다.
- `getcwd`와 `strrchr`을 사용하여 **현재 디렉토리의 마지막 경로명**을 추출하여 표시합니다.

3. 기본 명령 실행 (fork/exec/wait + 에러처리)

```
void execute_external_command(char **argv, int is_background, char *infil
e, char *outfile) {
    pid_t pid;
```

```
    int status;

    // 자식 프로세스 생성
    if ((pid = fork()) == 0) {
        // **자식 프로세스**

        // 1. 리다이렉션 처리
        if (infile != NULL) {
            // 입력 리다이렉션 (<)
            int fd_in = open(infile, O_RDONLY);
            if (fd_in < 0) {
                fprintf(stderr, "xsh: No such file or directory: %s\n", infile);
                exit(1);
            }
            // 표준 입력 (STDIN_FILENO)을 파일 디스크립터로 대체
            dup2(fd_in, STDIN_FILENO);
            close(fd_in);
        }

        if (outfile != NULL) {
            // 출력 리다이렉션 (>)
            // 파일이 없으면 생성, 있으면 덮어쓰기 (O_TRUNC), 쓰기 전용 (O_WRONL
Y)
            int fd_out = open(outfile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
            if (fd_out < 0) {
                perror("xsh: failed to open output file");
                exit(1);
            }
            // 표준 출력 (STDOUT_FILENO)을 파일 디스크립터로 대체
            dup2(fd_out, STDOUT_FILENO);
            close(fd_out);
        }

        // 2. 외부 명령 실행
        // PATH 환경 변수에서 실행 파일을 찾음 (execvp)
        execvp(argv[0], argv);

        // execvp가 실패했을 경우 (명령어를 찾지 못했거나 실행 권한이 없는 경우)
```

```
        fprintf(stderr, "xsh: command not found: %s\n", argv[0]);
        exit(1); // 자식 프로세스 종료
    } else if (pid > 0) {
        // **부모 프로세스**

        if (is_background) {
            // 백그라운드 실행 (&)
            // 보너스: 시작 메시지를 노란색으로 출력
            printf(ANSI_COLOR_YELLOW "[bg] started pid=%d\n" ANSI_COLOR
_RESET, pid);
            // 쉘은 기다리지 않고 프롬프트로 즉시 복귀
        } else {
            // 포그라운드 실행
            // 자식 프로세스가 끝날 때까지 대기
            do {
                waitpid(pid, &status, 0); // 0 옵션으로 블록 (대기)
            } while (!WIFEXITED(status) && !WIFSIGNALED(status));
        }
    } else {
        // fork 실패 시
        perror("fork error");
    }
}
```

- `execute_external_command` 함수에서 처리합니다.
- `fork()`를 호출하여 **자식 프로세스를 생성**하고, 부모 프로세스는 대기합니다.
- 자식 프로세스에서는 `execvp()`를 호출하여 **외부 명령을 실행**합니다.
- `execvp()`가 실패하면 "xsh: command not found: [명령어]"를 출력하고 종료합니다.
- 부모 프로세스는 `waitpid()`를 사용하여 **자식 프로세스가 종료될 때까지 대기**합니다.

4. 내장 명령어 (exit, cd, pwd)

```
int execute_builtin_command(char **argv) {
    if (strcmp(argv[0], "exit") == 0) {
        // 'exit' 명령어: 쉘 종료
```

```
        exit(0);
    } else if (strcmp(argv[0], "pwd") == 0) {
        // 'pwd' 명령어: 현재 작업 디렉토리 출력
        char cwd[1024];
        if (getcwd(cwd, sizeof(cwd)) != NULL) {
            printf("%s\n", cwd);
        } else {
            perror("pwd error");
        }
        return 1;
    } else if (strcmp(argv[0], "cd") == 0) {
        // 'cd' 명령어: 디렉토리 변경
        char *target_dir = argv[1];
        if (target_dir == NULL) {
            // 인자가 없으면 HOME 디렉토리로 이동
            target_dir = getenv("HOME");
        }

        if (target_dir != NULL && chdir(target_dir) != 0) {
            // 디렉토리 변경 실패 시 에러 출력
            perror("cd error");
        }
        return 1;
    }

    return 0; // 내장 명령어가 아님
}
```

- `execute_builtin_command` 함수에서 `strcmp`를 사용하여 **`exit`**, **`cd`**, **`pwd`** 명령어를 직접 처리합니다.
- `cd`는 `chdir()`를 사용하며, 인자가 없으면 환경 변수 `$HOME`으로 이동합니다.
- `pwd`는 `getcwd()`를 사용하여 현재 경로를 출력합니다.
- 내장 명령어를 실행했으면 1을 반환하여 외부 명령어 실행을 건너뛰게 합니다.

5. 리다이렉션 (<, >)

```
int parse_command(char *line, char **argv, int *is_background, char **infil
e, char **outfile) {
    int argc = 0;
    char *token;
    char *saveptr;

    // 공백을 기준으로 토큰 분리
    token = strtok_r(line, " \t", &saveptr);
    while (token != NULL) {
        if (strcmp(token, "&") == 0) {
            // 백그라운드 실행 기호 (&)
            *is_background = 1;
        } else if (strcmp(token, "<") == 0) {
            // 입력 리다이렉션 (<)
            token = strtok_r(NULL, " \t", &saveptr);
            if (token) *infile = token;
        } else if (strcmp(token, ">") == 0) {
            // 출력 리다이렉션 (>)
            token = strtok_r(NULL, " \t", &saveptr);
            if (token) *outfile = token;
        } else {
            // 일반 명령어 인자
            argv[argc++] = token;
        }

        // 다음 토큰 가져오기
        token = strtok_r(NULL, " \t", &saveptr);
    }

    argv[argc] = NULL; // execvp를 위해 인자 리스트의 끝은 NULL로 설정
    return argc;
}
```

- `parse_command` 함수에서 `<`와 `>` 기호 뒤의 파일명을 파싱하여 `infile`과 `outfile` 변수에 저장합니다.
- `execute_external_command`의 자식 프로세스에서 `open()` 함수와 `dup2()` 함수를 사용하여 **표준 입/출력을 해당 파일로 대체**합니다.

- 입력 (`<`): `O_RDONLY`로 열고 `STDIN_FILENO`로 `dup2`.
- 출력 (`>`): `O_WRONLY | O_CREAT | O_TRUNC`로 열고 `STDOUT_FILENO`로 `dup2`.

6. 환경 변수 출력 ($VAR)

```
void expand_environment_variables(char **argv) {
    for (int i = 0; argv[i] != NULL; i++) {
        if (argv[i][0] == '$') {
            // '$' 다음 문자열(변수명)을 가져와 환경 변수 값을 찾음
            char *var_name = argv[i] + 1;
            char *value = getenv(var_name);

            if (value != NULL) {
                // 환경 변수 값이 존재하면 그 값으로 대체
                argv[i] = value;
            } else {
                // 환경 변수가 정의되지 않은 경우 빈 문자열로 처리하거나 그대로 둠
            }
        }
    }
}
```

- `expand_environment_variables` 함수에서 인자 배열을 순회하며 `$변수명` **패턴을 찾습니다.**
- `getenv()`를 사용하여 해당 환경 변수의 **값을 얻어와 인자를 대체**합니다.

실행 결과


문제
출력
디버그 콘솔
터미널
포트
bash - answer
mkxvi316@mkxvi316:~/aisys/Assignment/hw3/answer$ ./xsh
xsh[29058]:answer> ls
hello.txt out.txt sleep_print.sh sorted_out.txt xsh xsh.c
xsh[29058]:answer> foobar
xsh: command not found: foobar
xsh[29058]:answer> cd ..
xsh[29058]:hw3> pwd
/home/mkxvi316/aisys/Assignment/hw3
xsh[29058]:hw3> cd answer
xsh[29058]:answer> echo "Hello world" > hello.txt
xsh[29058]:answer> cat < hello.txt
"Hello world"
xsh[29058]:answer> echo $HOME
/home/mkxvi316
xsh[29058]:answer> echo $PATH
xsh[29058]:answer> exit
mkxvi316@mkxvi316:~/aisys/Assignment/hw3/answer$ ./xsh
xsh[29445]:answer> ls > out.txt
xsh[29445]:answer> cat out.txt
hello.txt
out.txt
sleep_print.sh
sorted_out.txt
xsh
xsh.c
xsh[29445]:answer> sort < out.txt > sorted_out.txt
xsh[29445]:answer> cat sorted_out.txt
hello.txt
out.txt
sleep_print.sh
sorted_out.txt
xsh
xsh.c
xsh[29445]:answer> sh sleep_print.sh
Hi~
xsh[29445]:answer> sh sleep_print.sh &
[bg] started pid=29759
xsh[29445]:answer> Hi~
xsh[29445]:answer> exit
mkxvi316@mkxvi316:~/aisys/Assignment/hw3/answer$

✅ **성공!**