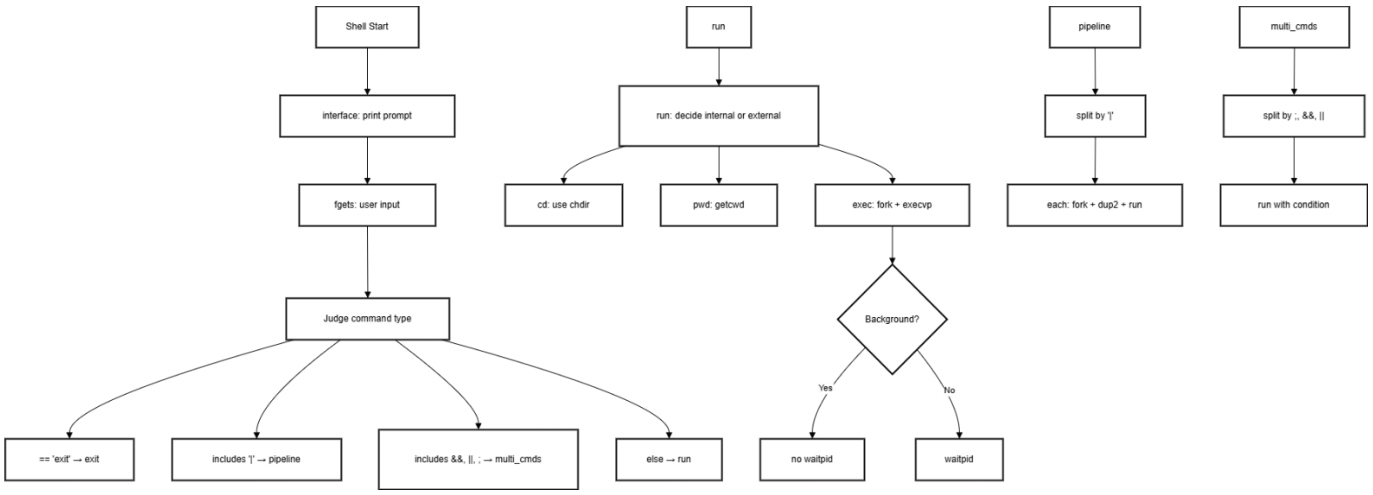
2024350030 신창헌

1. 전체 로직



2. 함수 설명

1) 현재 디렉토리 표시 및 bash 스타일 프롬프트

ㄱ. interface()

```
void interface(void)
{
    char cwd[1024];                         // (1) 현재 디렉토리 저장용 버퍼
    char *usrname = getenv("USER");         // (2) 환경변수에서 사용자 이름 가져오기
    char hostname[1024];
    getcwd(cwd, sizeof(cwd));               // (3) 현재 디렉토리 경로 얻기
    gethostname(hostname, sizeof(hostname)); // (4) 현재 호스트 이름 얻기

    if (usrname != NULL)
    {
        printf("%s@%s:%s$", usrname, hostname, cwd);   // (5) user@host:/path$ 형태 출
력
    }
}
```

2) cd, pwd를 직접 구현

ㄱ. cd(input)

```
int cd(char *input)
{
    if (strncpy(input, "cd", 3) == 0)           // (1) 명령어가 "cd"로 시작하는지 확인
    {
        char *path = input + 3;                 // (2) "cd " 다음 문자열을 경로로 처리
```

```
        if (chdir(path) != 0){                    // (3) 경로 이동 시도 → 실패 시
            perror("cd");                         // (4) 오류 메시지 출력
            return -1;
        } else return 0;                          // (5) 성공 시 0 반환
    }
    return -1;
}
```

ㄴ. pwd

```
int pwd(char *input)
{
    if (strcmp(input, "pwd") == 0){               // (1) 명령이 "pwd"인지 확인
        char cwd[1024];
        if(getcwd(cwd, sizeof(cwd))!=NULL){       // (2) 현재 경로 가져오기 성공 시
            printf("%s₩n", cwd);                  // (3) 경로 출력
            return 0;
        } else {
            perror("pwd");                        // (4) 실패 시 오류 출력
            return -1;
        }
    }
    return -1;
}
```

3) 파이프라인

ㄱ. pipeline(input)

```
char *token = strtok(input, "|");                 // (1) '|' 기준으로 명령어 분리
while(token != NULL && cmd_count < 10){
    cmds[cmd_count++] = token;                    // (2) 각 명령어를 배열에 저장
    token = strtok(NULL, "|");
}

int pipefd[2];
pipe(pipefd);                                     // (3) 파이프 생성

if(prev_fd != -1){
    dup2(prev_fd, 0);                             // (4) 이전 명령어의 출력을 현재 명령의
입력으로
}
...
dup2(pipefd[1], 1);                               // (5) 현재 명령어의 출력을 다음 명령어
입력으로 전달
```

4) 다중명령어 ;, &&, ||

ㄱ. multi_cmds(input, bkg)

```
char *next_and = strstr(input, "&&");
char *next_or = strstr(input, "||");
char *next_seq = strstr(input, ";");          // (1) 구문자 탐색

if (next) *next = '₩0';                       // (2) 현재 명령어만 잘라냄
...
last_status = run(input, bkg);                // (3) 실행 후 결과값 저장

if(type==1 && last_status==-1) break;         // (4) && 실패 시 종료
if(type==2 && last_status==0) break;          // (5) || 성공 시 종료
```

5) 백그라운드 실행

ㄱ. check_bkg(input)

```
if(input[strlen(input)-1]=='&'){              // (1) 문자열 끝에 & 있는지 확인
    background = -1;
    input[strlen(input)-1]='₩0';              // (2) & 문자 제거
    input[strcspn(input, " ₩t₩r₩n")]=0;       // (3) 끝 공백 제거
}
```

이후 exec으로 전달

ㄴ. exec(input, bkg)

```
if(!bkg) waitpid(pid, NULL, 0);               // (1) 포그라운드 실행 시 자식 종료까지 대기
else printf("백그라운드 실행 : pid=%d₩n", pid); // (2) 백그라운드이면 바로 출력
```

6) exit 입력 시 종료

```
if (strcmp(input, "exit") == 0) break;        // (main.c 내부)
```

etc...

run(input, bkg)

```
if(strncmp(input, "cd ", 3) == 0){
    return cd(input);
}else if(strcmp(input, "pwd") == 0){
    return pwd(input);
}else {
    return exec(input, bkg);
}
```