# Assignment5 : AI Assist Shell

**[AI기반시스템프로그래밍]**

작성일 : 2025.12.23.

학번 : 23013393

작성자 : 최지혁

---

# Assignment5 : AI Assist Shell

> 이 과제는 **"Google Gemini Pro : 빠른 모드"**를 참고하여 작성하였습니다.

**코드**

**[common.h]**

```
#ifndef COMMON_H
#define COMMON_H
#include <semaphore.h>

#define SHM_NAME "/ai_shm"
#define SEM_REQ "/sem_req"
#define SEM_RES "/sem_res"
#define MAX_BUF 4096

typedef struct {
    char question[MAX_BUF];
    char answer[MAX_BUF];
} shm_data;
#endif
```

**[ai_shell.c]**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <termios.h>
#include <time.h>
#include "common.h"

int current_mode = 0;
shm_data *shared_mem;
sem_t *sem_req, *sem_res;

void set_conio_mode(struct termios *old_t) {
    struct termios new_t;
    tcgetattr(STDIN_FILENO, old_t);
    new_t = *old_t;
    new_t.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &new_t);
}

void reset_conio_mode(struct termios *old_t) {
    tcsetattr(STDIN_FILENO, TCSANOW, old_t);
}

void execute_system_cmd(char *cmd) {
    if (strlen(cmd) == 0) return;
    char *args[64];
    int i = 0;
    char *token = strtok(cmd, " ");
    while (token) { args[i++] = token; token = strtok(NULL, " "); }
    args[i] = NULL;
```

```
    if (strcmp(args[0], "exit") == 0) exit(0);
    if (strcmp(args[0], "pwd") == 0) {
        char cwd[1024]; getcwd(cwd, sizeof(cwd)); printf("%s\n", cwd);
        return;
    }
    if (strcmp(args[0], "cd") == 0) {
        if (args[1] == NULL || chdir(args[1]) != 0) perror("cd failed");
        return;
    }

    if (fork() == 0) {
        if (execvp(args[0], args) == -1) printf("command not found\n");
        exit(1);
    } else wait(NULL);
}

int main() {
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    ftruncate(shm_fd, sizeof(shm_data));
    shared_mem = mmap(0, sizeof(shm_data), PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);

    sem_req = sem_open(SEM_REQ, O_CREAT, 0666, 0);
    sem_res = sem_open(SEM_RES, O_CREAT, 0666, 0);

    struct termios old_t;
    char input[MAX_BUF];
    int idx = 0;
    memset(input, 0, MAX_BUF);

    printf("AI Assist Shell. Press Ctrl+T to switch mode.\n");
    while (1) {
        printf("\r%s> %s", current_mode ? "\x1b[35mAI\x1b[0m" : "Shell", inpu
t);
        fflush(stdout);

        set_conio_mode(&old_t);
        char c = getchar();
```

```
        reset_conio_mode(&old_t);

        if (c == 20) { // Ctrl + T
            current_mode = !current_mode;
            memset(input, 0, MAX_BUF); idx = 0;
            printf("\n[Mode Switched to %s]\n", current_mode ? "AI" : "Shell");
            continue;
        } else if (c == '\n' || c == '\r') {
            input[idx] = '\0';
            printf("\n");
            if (current_mode == 0) {
                execute_system_cmd(input);
            } else {
                memset(shared_mem->answer, 0, MAX_BUF);
                strcpy(shared_mem->question, input);
                sem_post(sem_req);
                printf("\x1b[36m🤖[AI] Waiting for response...\x1b[0m\n");

                struct timespec ts;
                clock_gettime(CLOCK_REALTIME, &ts);
                ts.tv_sec += 600; // 10분 타임아웃 적용

                if (sem_timedwait(sem_res, &ts) == -1) {
                    printf("[AI] 10분간 응답이 없어 질의를 무시합니다.\n");
                } else {
                    printf("[AI] %s\n", shared_mem->answer);
                }
            }
            memset(input, 0, MAX_BUF); idx = 0;
        } else if (c == 127 || c == 8) { // Backspace
            if (idx > 0) { input[--idx] = '\0'; printf("\b \b"); }
        } else {
            if (idx < MAX_BUF - 1) input[idx++] = c;
        }
    }
    return 0;
}
```

**[ai_helper.c]**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include "common.h"

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) { perror("shm_open failed"); exit(1); }
    shm_data *shared_mem = mmap(0, sizeof(shm_data), PROT_READ | PRO
T_WRITE, MAP_SHARED, shm_fd, 0);
    sem_t *sem_req = sem_open(SEM_REQ, 0);
    sem_t *sem_res = sem_open(SEM_RES, 0);

    // 요청하신 프롬프트 적용
    char sys_p[1024] = "너는 우분투 전문가야. 우분투 명령어 설명을 한국어로 답변
해. 그리고 간결하게 답변해.";

    printf("AI Helper (gemma3:1b) running...\n");

    while (1) {
        sem_wait(sem_req);
        printf("[Log] 질문 수신: %s\n", shared_mem->question);
        memset(shared_mem->answer, 0, MAX_BUF);

        char cmd[MAX_BUF * 2];
        snprintf(cmd, sizeof(cmd),
                "curl -s http://localhost:11434/api/generate -d '{"
                "\"model\": \"gemma3:1b\","
                "\"prompt\": \"%s 사용자의 질문: %s\","
                "\"stream\": false"
                "}' | python3 -c \"import sys, json; data=json.load(sys.stdin); prin
t(data.get('response', '응답 생성 실패'))\"",
```

```
            sys_p, shared_mem→question);

        FILE *fp = popen(cmd, "r");
        if (fp) {
            char response[MAX_BUF] = "";
            char buffer[512];
            while (fgets(buffer, sizeof(buffer), fp)) {
                if (strlen(response) + strlen(buffer) < MAX_BUF - 50) strcat(respo
nse, buffer);
            }
            strcat(response, "\n<<<END>>>");
            strncpy(shared_mem→answer, response, MAX_BUF - 1);
            pclose(fp);
            printf("[Log] 응답 완료\n");
        }
        sem_post(sem_res);
    }
    return 0;
}
```

---

# 코드 구현

## 1. “common.h”

```
#ifndef COMMON_H
#define COMMON_H
#include <semaphore.h>

#define SHM_NAME "/ai_shm"
#define SEM_REQ "/sem_req"
#define SEM_RES "/sem_res"
#define MAX_BUF 4096

typedef struct {
    char question[MAX_BUF];
    char answer[MAX_BUF];
```

```
} shm_data;
#endif
```

이 파일은 독립된 두 프로세스(`ai_shell`, `ai_helper`)가 **데이터를 공유하고 동기화**하기 위해 사용하는 공통 규격입니다.

- **IPC 자원 이름 정의**: 공유 메모리(`SHM_NAME`)와 세마포어(`SEM_REQ`, `SEM_RES`)의 경로 이름을 통일하여 서로 동일한 자원에 접근하게 합니다.
- **공유 데이터 구조체(`shm_data`)**:
  - `question`: 사용자의 질문을 저장하여 Helper에게 전달합니다.
  - `answer`: AI의 답변을 저장하여 Shell에게 전달합니다.
  - `MAX_BUF (4096)`: 넉넉한 버퍼 크기를 지정하여 긴 명령어나 답변도 처리 가능하게 합니다.

## 2. 모드 변경

```
if (c == 20) { // Ctrl + T
        current_mode = !current_mode;
    memset(input, 0, MAX_BUF); idx = 0;
    printf("\n[Mode Switched to %s]\n", current_mode ? "AI" : "Shell");
    continue;
```

- `termios` 설정을 통해 터미널을 비정규 모드로 전환하여 엔터 입력 없이도 **Ctrl + T** 키를 즉시 감지하고 쉘 모드와 AI 모드 사이를 전환하도록 구현했습니다.

## 3. IPC 기법 구현

```
// 공유 메모리 생성 및 연결
int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
shared_mem = mmap(0, sizeof(shm_data), PROT_READ | PROT_WRITE, MA
P_SHARED, shm_fd, 0);

// 세마포어 생성
```

```
sem_req = sem_open(SEM_REQ, O_CREAT, 0666, 0);
sem_res = sem_open(SEM_RES, O_CREAT, 0666, 0);
```

- **POSIX Shared Memory**와 **Semaphore**를 사용하여 두 프로세스 간 통신을 구현했습니다. `sem_req`는 질문 요청 시그널로, `sem_res`는 답변 완료 시그널로 사용하여 데이터 동기화를 보장합니다.

## 4. 일반쉘 - 내장 명령어 구현

```
if (strcmp(args[0], "exit") == 0) exit(0);
if (strcmp(args[0], "pwd") == 0) {
    char cwd[1024]; getcwd(cwd, sizeof(cwd)); printf("%s\n", cwd);
    return;
}
if (strcmp(args[0], "cd") == 0) {
    if (args[1] == NULL || chdir(args[1]) != 0) perror("cd failed");
    return;
}
```

- 시스템 호출 함수(`chdir`, `getcwd`)를 활용하여 쉘의 필수 내장 명령어인 **exit, cd, pwd**를 구현했습니다.

## 5. 일반쉘 - 기본 명령 구현

```
if (fork() == 0) {
    if (execvp(args[0], args) == -1) printf("command not found\n");
    exit(1);
} else wait(NULL);
```

- `fork()`를 통해 자식 프로세스를 생성하고 `execvp()`를 실행하여 외부 프로그램을 수행합니다. 명령어가 존재하지 않을 경우 요구사항대로 `command not found`를 출력합니다.

## 6. AI모드 - 한국어로 질의시 정확한 우분투 명령어 제시

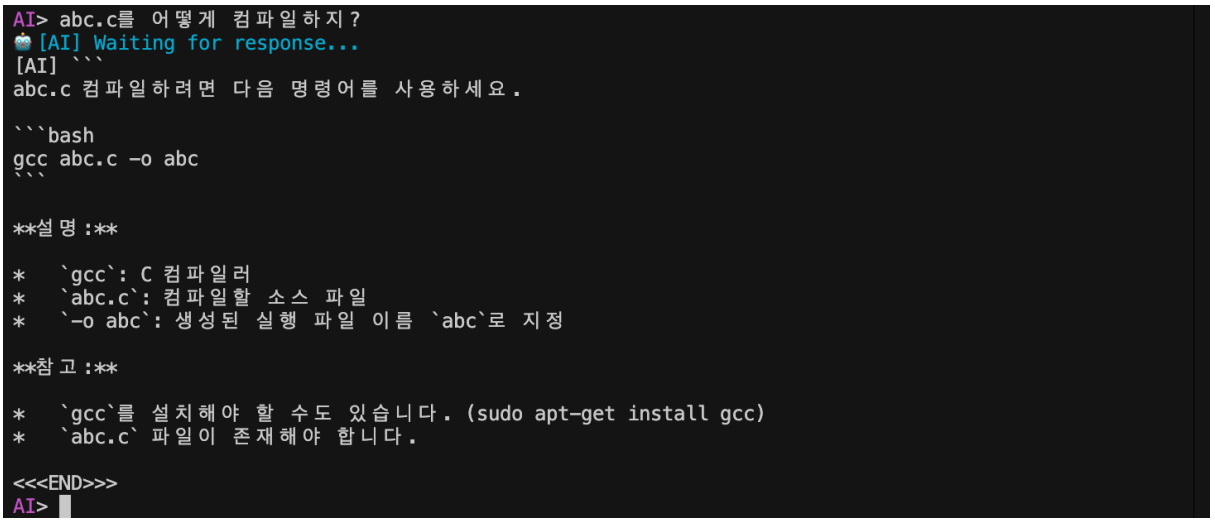
```
// ai_helper.c: gemma3:1b 호출 및 마커 추가
snprintf(cmd, ..., "model: gemma3:1b, prompt: %s ...", sys_p, shared_mem->question);
strcat(response, "\n<<<END>>>");

// ai_shell.c: 10분 타임아웃
ts.tv_sec += 600; // 600초 = 10분
if (sem_timedwait(sem_res, &ts) == -1) printf("[AI] 10분간 응답이 없어 질의를 무시합니다.\n");
```

- **gemma3:1b** 모델을 사용하여 우분투 관련 질의에 답변합니다. 응답의 끝을 알리는 **<<<END>>>** 마커를 포함하며, **10분** 타임아웃을 적용하여 무한 대기를 방지했습니다.

---

## 실행 결과



```
AI> 우분투의 cp 명령어 잘 쓰는 옵션 좀 알려줘
[AI] Waiting for response...
[AI] ## 우분투 cp 명령어 활용 가이드 (간단 버전)

**cp** 명령어는 파일 또는 디렉토리의 내용을 다른 곳으로 복사하는 데 사용됩니다.

**기본 사용법:**

*   `cp [원본 파일] [대상 파일]`: 원본 파일의 내용을 대상 파일로 복사합니다.
*   `cp -r [원본 디렉토리] [대상 디렉토리]`: 원본 디렉토리의 내용을 대상 디렉토리로 복사합니다.

**주요 옵션:**

*   `-r` (recursive): 디렉토리 전체를 복사합니다.
*   `-i` (interactive): 복사 중 오류 발생 시 오류 메시지를 표시합니다.
*   `-f` (force): 복사 중 오류 발생 시 수정 없이 복사합니다.
*   `-v` (verbose): 복사 과정의 정보를 표시합니다.

**예시:**

*   `cp my_file.txt /home/user/backup/` (my_file.txt를 /home/user/backup/에 복사)
*   `cp -r my_folder /home/user/documents` (my_folder를 /home/user/documents로 복사)

궁금한 점이 있으면 언제든지 질문해주세요.

<<<END>>>
AI>
```

```
AI> 우분투에서 압축은 어떻게 하지?
[AI] Waiting for response...
[AI] 우분투에서 압축은 다음과 같습니다.

1. **`tar` 명령:** 파일들을 묶어 압축 파일을 만듭니다. (`tar -czf - directory/file/`)
2. **`gzip` 명령:** 파일들을 압축합니다. (`gzip -c -d directory/file/`)
3. **`zip` 명령:** 파일들을 압축하여 zip 파일로 만듭니다. (`zip -r -d directory/zipfile.zip`)

**간단하게:**

*   `tar` 압축, `gzip` 압축, `zip` 압축 순서.

궁금한 점이 있다면 질문해주세요.

<<<END>>>
AI>
```

```
AI> 실시간으로 성능 보는 명령어가 뭐였지?
[AI] Waiting for response...
[AI] `top`

**참고:** `top` 명령어는 현재 시스템의 리소스 사용량, 프로세스 정보 등을 실시간으로 보여줍니다.

<<<END>>>
AI>
```

```
AI> 현재 위치를 확인하는 명령어가 뭐였지?
[AI] Waiting for response...
[AI] `pwd` 입니다.

<<<END>>>
AI>
```

```
AI> vi 관련 명령어 좀 알려줘
🤖[AI] Waiting for response...
[AI] ## 우분투 vi 명령어 설명 (간결 버전)

**1. 텍스트 편집:**

*   **`vi`**: 기본 텍스트 편집기. 마우스로 텍스트를 변경하고, `i` 키로 텍스트를 수정.
*   **`v`**: `vi`의 기본 모드. 텍스트를 변경할 때 사용.
*   **`q`**: `vi` 종료.
*   **`Esc`**: `vi` 종료.

**2. 특정 위치 텍스트 변경:**

*   **`edit <파일명>`**: `vi`에서 지정된 파일의 텍스트를 변경.
*   **`x <파일명>`**: `vi`에서 지정된 파일의 텍스트를 삭제.
*   **`n`**: 다음 줄에 이어지는 새로운 줄을 추가.
*   **`p`**: 현재 줄의 앞/뒤 1 줄을 삭제.
*   **`D`**: 현재 줄의 끝에 있는 단어 삭제.
*   **`u`**: 현재 줄의 끝에 있는 단어 추가.
*   **`yy`**: 현재 줄의 내용을 삭제.
*   **`yw`**: 현재 줄의 내용을 수정.

**3. 기타:**

*   **`set`**: `vi` 설정 (예: 줄 1에서 시작, 자동 저장 등).
*   **`help`**: `vi` 도움말 보기.

**참고:** `vi`는 문법이 약한 텍스트 편집기이므로, 문법에 맞지 않는 텍스트를 변경하면 `vi`가 자동으로 종료됩니다.

궁금한 점이 있다면 언제든지 질문해주세요.

<<<END>>>
AI>
```

```
AI> 프로세스 관련 명령어 좀 알려줘
🤖[AI] Waiting for response...
[AI] 알겠습니다. 우분투 프로세스 관련 명령어는 다음과 같습니다.

**기본 명령어:**

*   **`ps`**: 현재 실행 중인 프로세스 목록 보기
*   **`top`**: 시스템 전체 CPU 및 메모리 사용률, 프로세스 목록 보기
*   **`htop`**: `top`보다 더 자세한 정보 제공 (선택 사항)
*   **`kill`**: 프로세스 종료 (예: `kill -9 <프로세스_ID>`)
*   **`killall`**: 특정 프로세스 종료 (예: `killall firefox`)
*   **`df`**: 디스크 공간 사용량 확인

**추가 명령어:**

*   **`ps aux`**: 모든 사용자에게 실행 중인 모든 프로세스 상세 정보 보기
*   **`grep`**: 특정 패턴이 포함된 줄 찾기 (예: `grep "python" /var/log/`)
*   **`wc`**: 줄 번호, 줄 길이, 문자 수 등 정보 보기

**참고:**

*   **프로세스 ID (PID)**를 통해 특정 프로세스 종료 가능
*   **`man <명령어>`**: 명령어 상세 사용법 확인

궁금한 점이 있다면 언제든지 질문해주세요.

<<<END>>>
AI>
```

```
AI> 권한 관련 명령어도 알려줘
[AI] Waiting for response...
[AI] 알겠습니다. 우분투 명령어 사용법에 대한 한국어 설명과 간결한 답변을 드리겠습니다.

**기본 명령어:**

*   **`man <command>`**: 명령어 상세 사용법 안내 (예: `man ls`)
*   **`pwd`**: 현재 작업 디렉토리 확인
*   **`ls`**: 파일 및 디렉토리 목록 보기
*   **`cd`**: 디렉토리 변경
*   **`mkdir <directory>`**: 디렉토리 생성
*   **`rm <file>`**: 파일 삭제
*   **`cp <source> <destination>`**: 파일 복사
*   **`mv <source> <destination>`**: 파일 이동
*   **`rmdir <directory>`**: 디렉토리 삭제 (디렉토리가 비어있어야 함)
*   **`touch <file>`**: 파일 생성 (파일이 없으면 생성)
*   **`cat <file>`**: 파일 내용 출력

**추가 도움말:**

*   **`help`**: 자동으로 제공되는 도움말 (일반적인 명령어)
*   **`--help`**: 명령어 상세 사용법

**참고:**

*   **`sudo`**: 관리자 권한으로 명령 실행 (필요한 경우)

궁금한 점이 있다면 언제든지 질문해주세요.

<<<END>>>
AI>
```

```
AI> 디렉토리 관련 명령어를 알려줘
[AI] Waiting for response...
[AI] ## 우분투 명령어 설명 (간결 버전)

**기본 명령어:**

*   **ls:** 현재 디렉토리의 파일 및 디렉토리 목록 보기
*   **cd:** 디렉토리 변경
*   **pwd:** 현재 위치의 경로 보기
*   **mkdir:** 새 디렉토리 생성
*   **rm:** 파일 삭제
*   **rmdir:** 빈 디렉토리 삭제
*   **touch:** 빈 파일 생성
*   **cp:** 파일 복사
*   **mv:** 파일 이동 및 이름 변경

**추가:**

*   **find:** 파일 검색
*   **grep:** 파일 내용 검색

**참고:**

*   **bash:** 우분투에서 주로 사용되는 쉘.

**궁금한 내용이 있다면, 구체적인 질문을 해주세요.**

<<<END>>>
AI>
```

```
AI> 우분투에서 파일 내용 확인은 어떻게 하더라?
[AI] Waiting for response...
[AI] 우분투에서 파일 내용 확인 방법은 다음과 같습니다.

1.  **`cat`**: 파일 내용을 보여줍니다. (예: `cat my_file.txt`)
2.  **`less`**: 파일 내용을 페이지 단위로 볼 수 있게 해줍니다. (예: `less my_file.txt`)
3.  **`head`**: 파일의 맨 앞부분을 보여줍니다. (예: `head my_file.txt`)
4.  **`tail`**: 파일의 맨 끝부분을 보여줍니다. (예: `tail my_file.txt`)
5.  **`grep`**: 특정 패턴의 문서를 찾습니다. (예: `grep "keyword" my_file.txt`)
6.  **`vim`**: 텍스트 편집기입니다. 파일 내용을 수정하고 편집할 수 있습니다. (예: `vim my_file.txt`)

더 궁금한 점이 있으시면 언제든지 질문해주세요.

<<<END>>>
AI>
```

✅ **성공!**