뽀모도로 캘린더

시스템 프로그래밍 5조

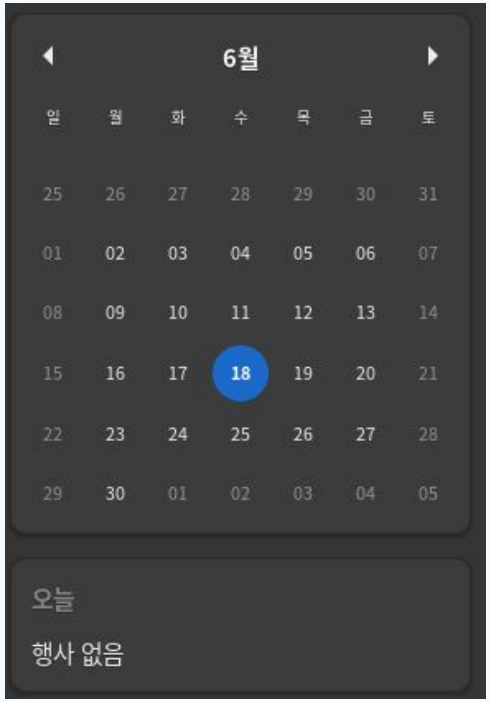
송민우, 김건희, 최원서, 김재혁, 이채민, 손민승

Table of Contents

1. Introduction

아이디어

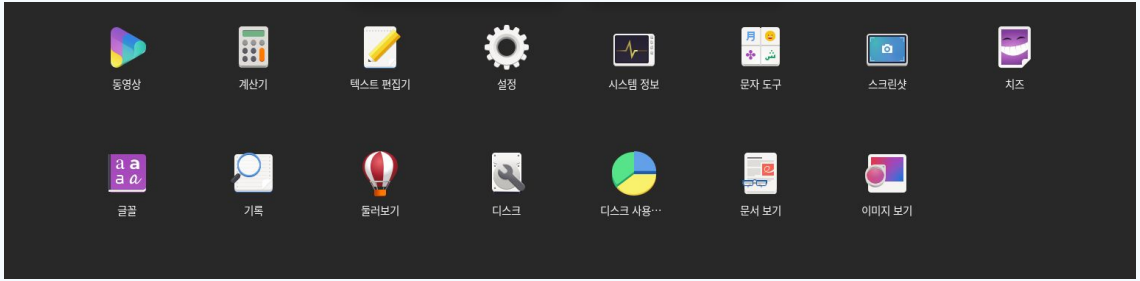
로키리눅스 기본 캘린더



로키 리눅스 일정확인은 가능

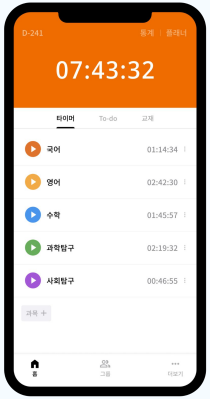
간편하게 일정 추가 X

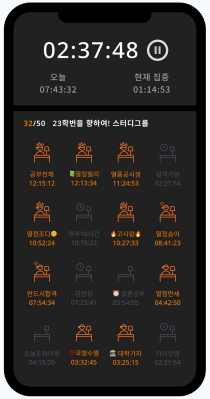
로키리눅스 기본 프로그램에 타이머 기능 X



1. Introduction

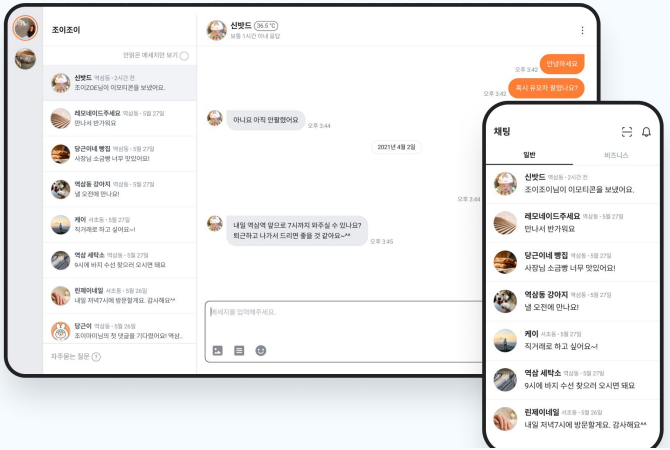
아이디어





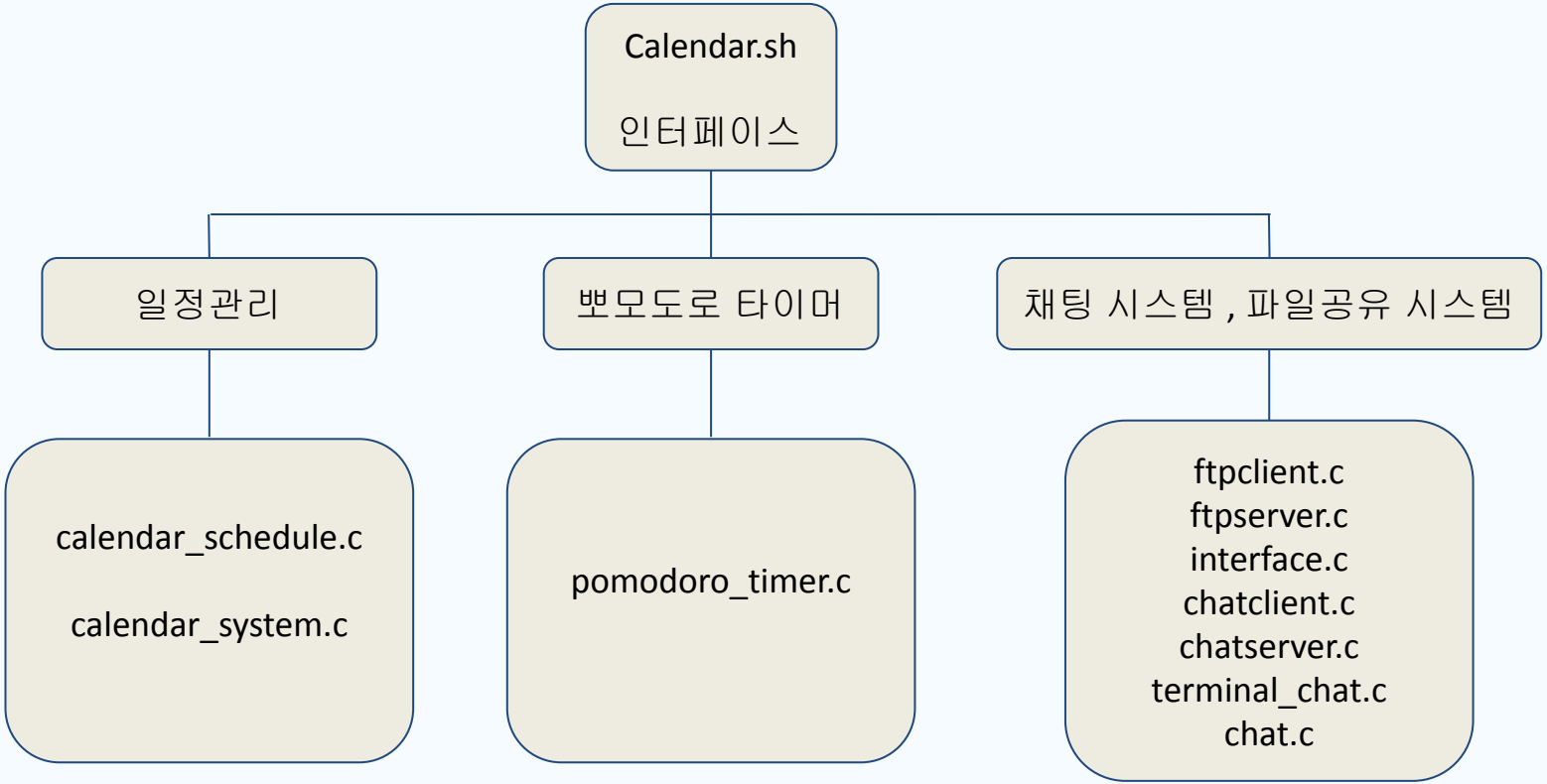
‘열품타’ 에 영감을 얻어 시스템을 설계·구현

터미널 기반
뽀모도로 타이머+ 캘린더 + 채팅기능 구현



1. Introduction

코드 전체 구성



2. Interface

Makefile

Makefile

```
# [Makefile 수정본2.txt 기반 코드 시작]
CC = gcc
CFLAGS = -Wall -Wextra -std=c99 -O2
```

```
calendar_system: calendar_system.c
	@echo "🔨 calendar_system 컴파일 중..."
	@$(CC) $(CFLAGS) -o calendar_system calendar_system.c

calendar_schedule: calendar_schedule.c
	@echo "🔨 calendar_schedule 컴파일 중..."
	@$(CC) $(CFLAGS) -o calendar_schedule calendar_schedule.

# 터미널 채팅 프로그램 컴파일 규칙
terminal_chat: terminal_chat.c
	@echo "🔨 terminal_chat 컴파일 중..."
	@$(CC) $(CFLAGS) -o terminal_chat terminal_chat.c

# 채팅 서버/클라이언트 컴파일 규칙
chatserver: chatserver.c
	@echo "🔨 chatserver 컴파일 중..."
	@$(CC) $(CFLAGS) -o chatserver chatserver.c

chatclient: chatclient.c
	@echo "🔨 chatclient 컴파일 중..."
	@$(CC) $(CFLAGS) -o chatclient chatclient.c

# 뽀모도로 타이머 컴파일 규칙
pomodoro_timer: pomodoro_timer.c
	@echo "🔨 pomodoro_timer 컴파일 중..."
	@$(CC) $(CFLAGS) -o pomodoro_timer pomodoro_timer.c
```

gcc , 컴파일 하는 코드를 변수로 지정해

컴파일 코드를

CC, CFLAG 변수에 지정

각 파일 컴파일 시 호출해 코드 효율성 개선

2. Interface

Makefile

Makefile

```
# 실행 파일
TARGETS = calendar_system calendar_schedule terminal_chat pomodoro_timer ftpserver ftpclient chatserver chatclient

# 기본 빌드
all: $(TARGETS)
	@echo "✅ 모든 프로그램 컴파일 완료!"
	@echo "📋 실행: ./calendar.sh"
	@chmod +x calendar.sh
	@touch schedules.txt
	@touch chat_log.txt
	@touch files.txt
```

```
[minseung@localhost project]$ make
🔨 calendar_system 컴파일 중...
🔨 calendar_schedule 컴파일 중...
🔨 terminal_chat 컴파일 중...
🔨 pomodoro_timer 컴파일 중...
gcc -Wall -Wextra -std=c99 -O2 -o ftpserver ftpserver.c
gcc -Wall -Wextra -std=c99 -O2 -o ftpclient ftpclient.c
🔨 chatserver 컴파일 중...
🔨 chatclient 컴파일 중...
✅ 모든 프로그램 컴파일 완료!
📋 실행: ./calendar.sh
```

```
[minseung@localhost project]$ ./calendar.sh
```

1. make 실행 후 상위 all 명령어 실행
2. 개별 컴파일 진행
3. calendar.sh 스크립트에 실행 권한 추가
4. 프로그램에 필요한 데이터 파일(빈 파일) 생성

schedules.txt : 일정 저장용

chat_log.txt : 채팅 로그용

files.txt : 파일공유시 파일정보 저장용

이후 ./calendar.sh 터미널에 입력

2. Interface

Main interface

2. Interface

Main interface

상단 UI

상단의 UI echo로 작성함

오른쪽 메뉴

EOF 명령어 사이에 오른쪽 메뉴 판을 작성해 menubox.txt에 저장

2. Interface

Main interface

```
PROGRAM_DIR=$(dirname "$0")
$PROGRAM_DIR/calendar_system $CURRENT_YEAR $CURRENT_MONTH > /tmp/calendar_output.txt
```

```c
int main(int argc, char *argv[]) {
        // 한글 지원 설정
        setlocale(LC_ALL, "ko_KR.UTF-8");

        int year, month;

        // 명령행 인자로 년/월 받기, 없으면 현재 날짜
        if (argc >= 3) {
                year = atoi(argv[1]);
                month = atoi(argv[2]);
        }
        else {
                get_today(&year, &month, NULL);
        }

        print_calendar(year, month);
        return 0;
```

달력 인터페이스 설정

1. calendar.sh이 있는 디렉터리 경로를 PROGRAM_DIR에 저장
2. calendar_system.c 바이너리 파일에 현재 연도, 월을 인자로 전달해 달력 인터페이스를 txt파일(calendar_output.txt)에 저장

2. Interface

Main interface

```
void print_calendar(int year, int month) {
        setlocale(LC_ALL, "ko_KR.UTF-8");

        int today_year, today_month, today_day;
        get_today(&today_year, &today_month, &today_day);

        int days = days_in_month(year, month);
        int first_day = get_day_of_week(year, month, 1);
        int total_width = TOTAL_CELLS * CELL_WIDTH + BORDER_COUNT;

        // 상단 테두리
        printf("┌");
        for (int i = 0; i < total_width - 2; i++) printf("─");
        printf("┐\n");

        // 헤더 (년월)
        printf("│");
        char header[20];
        sprintf(header, "%d년 %d월", year, month);
        int header_width = get_string_width(header);
        int header_spaces = (total_width - 2 - header_width) / 2;
        for (int i = 0; i < header_spaces; i++) printf(" ");
        printf("%s", header);
        for (int i = 0; i < total_width - 2 - header_spaces - header_width; i++) printf(" ");
        printf("│\n");

        // 요일 헤더
        printf("├");
        for (int i = 0; i < total_width - 2; i++) printf("─");
        printf("┤\n");

        printf("│");
        printf("%s", COLOR_RED);  print_fixed_cell("       일");  printf("%s", COLOR_RESET); printf("│");
                        print_fixed_cell("       월");  printf("│");
                        print_fixed_cell("       화");  printf("│");
                        print_fixed_cell("       수");  printf("│");
                        print_fixed_cell("       목");  printf("│");
                        print_fixed_cell("       금");  printf("│");
```

```
        printf("%s", COLOR_BLUE); print_fixed_cell("       토");  printf("%s", COLOR_RESET); printf("│\n");

        // 요일 구분선
        printf("├");
        for (int col = 0; col < 7; col++) {
                for (int i = 0; i < CELL_WIDTH; i++) printf("─");
                if (col < 6) printf("┼");
        }
        printf("┤\n");

        // 캘린더 본체 출력
        for (int week = 0; week < 6; week++) {
                printf("│");
                for (int weekday = 0; weekday < 7; weekday++) {
                        int cell_day = week * 7 + weekday - first_day + 1;
                        if (cell_day < 1 || cell_day > days) {
                                print_fixed_cell("");
                        }
                        else {
                                char day_str[30];  // 크기를 20에서 30으로 증가
                                if (year == today_year && month == today_month && cell_day == today_day) {
                                        sprintf(day_str, "       *%2d      ", cell_day);
                                        printf("%s%s%s", COLOR_BG_WHITE, day_str, COLOR_RESET);
                                        int remaining = CELL_WIDTH - 10;
                                        for (int i = 0; i < remaining; i++) printf(" ");
                                }
                                else if (weekday == 0) {
                                        sprintf(day_str, "       %2d", cell_day);
                                        printf("%s%s%s", COLOR_RED, day_str, COLOR_RESET);
                                        int remaining = CELL_WIDTH - 9;
                                        for (int i = 0; i < remaining; i++) printf(" ");
                                }
                                else if (weekday == 6) {
                                        sprintf(day_str, "       %2d", cell_day);
                                        printf("%s%s%s", COLOR_BLUE, day_str, COLOR_RESET);
                                        int remaining = CELL_WIDTH - 9;
                                        for (int i = 0; i < remaining; i++) printf(" ");
                                }
```

```
                                printf("%s%s%s", COLOR_BLUE, day_str, COLOR_RESET);
                                int remaining = CELL_WIDTH - 9;
                                for (int i = 1; i < remaining; i++) printf(" ");
                        }
                        else {
                                sprintf(day_str, "       %2d", cell_day);
                                print_fixed_cell(day_str);
                        }
                }
                if (weekday < 6) printf("│");
        }
        printf("│\n");

        // 일정 출
        printf("│");
        for (int weekday = 0; weekday < 7; weekday++) {
                int cell_day = week * 7 + weekday - first_day + 1;
                if (cell_day >= 1 && cell_day <= days) {
                        char title[50];
                        get_first_schedule(year, month, cell_day, title);
                        if (strlen(title) > 0) {
                                char formatted_title[60];
                                sprintf(formatted_title, "  %s", title);
                                print_fixed_cell(formatted_title);
                        }
                        else {
                                print_fixed_cell("");
                        }
                }
                else {
                        print_fixed_cell("");
                }
                if (weekday < 6) printf("│");
        }
        printf("│\n");

        // 빈 줄
        printf("│");
```

```
        // 빈 줄
        printf("│");
        for (int weekday = 0; weekday < 7; weekday++) {
                print_fixed_cell("");
                if (weekday < 6) printf("│");
        }
        printf("│\n");

        // 줄 구분선
        if (week < 5) {
                printf("├");
                for (int col = 0; col < 7; col++) {
                        for (int i = 0; i < CELL_WIDTH; i++) printf("─");
                        if (col < 6) printf("┼");
                }
                printf("┤\n");
        }
}

// 하단 테두리
printf("└");
for (int i = 0; i < total_width - 2; i++) printf("─");
printf("┘\n");
```

calendar_system.c

1. 오늘 날짜와 해당 월의 일수·1일 요일을 구합니다.
2. 상단 테두리와 “YYYY년 M월” 헤더를 가운데 정렬해 출력합니다.
3. 6주(42칸) 셀을 순회하며 오늘,일요일,토요일은 색상, 배경으로 강조하고 평일은 고정 폭 셀로 날짜를 그립니다.
4. 각 날짜 아래 첫 일정 제목과 빈 줄을 넣고 주간 구분선과 하단 테두리로 달력을 마무리합니다.

2. Interface

Main interface

```
# 일정 정보 업데이트
if [ -f "$PROGRAM_DIR/calendar_schedule" ]; then
        SCHEDULE_OUTPUT=$($PROGRAM_DIR/calendar_schedule show $CURRENT_YEAR $CURRENT_MONTH 2>/dev/null)
        SCHEDULE_RESULT=$?

        if [ $SCHEDULE_RESULT -eq 0 ] && [ ! -z "$SCHEDULE_OUTPUT" ]; then
                # 일정이 있는 경우 - 실제 일정 내용을 파싱하여 표시

                # 헤더 라인을 제거하고 실제 일정만 추출
                echo "$SCHEDULE_OUTPUT" | grep "일:" > /tmp/schedule_raw.txt

                # 최대 8개 일정을 메뉴박스에 표시할 수 있음
                LINE_COUNT=1
                while read -r schedule_line && [ $LINE_COUNT -le 8 ]; do
                        if [ ! -z "$schedule_line" ]; then
                                # 일정 라인을 메뉴박스 형식에 맞게 변환
                                # "일: 제목" 형식을 " 일일: 제목" 형식으로 변환하고 38자로 맞춤
                                FORMATTED_LINE=$(echo "$schedule_line" | sed 's/^ *\([0-9]\+\)일: \(.*\)/ \1일: \2/')

                                # 38자 길이로 맞추기 (한글 고려하여 간단하게 처리)
                                if [ ${#FORMATTED_LINE} -gt 36 ]; then
                                        FORMATTED_LINE=$(echo "$FORMATTED_LINE" | cut -c1-33)...
                                fi

                                # 남은 공간을 공백으로 채우기
                                # 화면 너비를 정확히 계산
                                # 가~힣 : 문자 너비 2, 영문/숫자/기호 : 문자 너비 1 -> 캘린더 레이아웃 유지를 위해 씀
                                REAL_WIDTH=$(echo "$FORMATTED_LINE" | awk 'BEGIN{w=0} {for(i=1;i<=length($0);i++){c=substr($0,i,1); w+=(c ~ /[가-힣]/) ? 2 : 1}} END{print w}')
                                SPACES_NEEDED=$((38 - REAL_WIDTH))
                                PADDING=""
                                for i in $(seq 1 $SPACES_NEEDED); do
                                        PADDING="$PADDING "
                                done

                                echo "|$FORMATTED_LINE$PADDING|" >> /tmp/schedule_lines.txt
                                LINE_COUNT=$((LINE_COUNT + 1))
                        else
                                break
                        fi
                done < /tmp/schedule_raw.txt

                # 빈 줄로 나머지 채우기
                while [ $LINE_COUNT -le 8 ]; do
                        echo "|                                      |" >> /tmp/schedule_lines.txt
                        LINE_COUNT=$((LINE_COUNT + 1))
                done
                rm -f /tmp/schedule_raw.txt
        else
                # 일정이 없는 경우
                echo "| 이번 달에는 일정이 없습니다.          |" > /tmp/schedule_lines.txt
                for i in $(seq 2 8); do
                        echo "|                                      |" >> /tmp/schedule_lines.txt
                done
        fi
        # 메뉴 박스의 일정 부분 (4-11번째 줄)을 새로운 일정 내용으로 교체
        {
                head -3 /tmp/menu_box.txt  # 헤더 부분
                cat /tmp/schedule_lines.txt  # 일정 부분
                tail -n +12 /tmp/menu_box.txt  # 메뉴 부분
        } > /tmp/menu_box_updated.txt
        cp /tmp/menu_box_updated.txt /tmp/menu_box.txt
        rm -f /tmp/schedule_lines.txt /tmp/menu_box_updated.txt
fi
# 캘린더와 메뉴를 좌우로 배치
paste -d ' ' /tmp/calendar_output.txt /tmp/menu_box.txt

echo
echo -e "${WHITE}현재 : ${BOLD}${CURRENT_YEAR}년 ${CURRENT_MONTH}월${RESET}
# 임시 파일 정리
rm -f /tmp/calendar_output.txt /tmp/menu_box.txt
```

메뉴 상단 일정 텍스트 설정

1. calendar.schedule.c 의 main함수에 인자를 줘 실행해 월별 일정 데이터 가져오기
2. 이미 캘린더에 설정된 일정 존재 여부 판별
3. 일정이 있을때 없을때 다른 패턴 텍스트를 추출
4. 추출한 텍스트를 schuedule_line.txt에 저장함

2. Interface

Main interface

```
        # 메뉴 박스의 일정 부분 (4-11번째 줄)을 새로운 일정 내용으로 교체
        {
                head -3 /tmp/menu_box.txt  # 헤더 부분
                cat /tmp/schedule_lines.txt  # 일정 부분
                tail -n +12 /tmp/menu_box.txt  # 메뉴 부분
        } > /tmp/menu_box_updated.txt
        cp /tmp/menu_box_updated.txt /tmp/menu_box.txt
        rm -f /tmp/schedule_lines.txt /tmp/menu_box_updated.txt
fi
# 캘린더와 메뉴를 좌우로 배치
paste -d ' ' /tmp/calendar_output.txt /tmp/menu_box.txt

echo
echo -e "${WHITE}현재 : ${BOLD}${CURRENT_YEAR}년 ${CURRENT_MONTH}월${RESET}
```

인터페이스 출력

1. 헤더부분은 앞에서 설정한 menubox.txt에서 텍스트를 추출함
2. 앞에서 설정한 schuedule_line.txt으로 일정부분 인터페이스 설정함
3. 두가지 결합된 텍스트를 menu_box.txt 저장
4. 달력(calendar_output.txt)과 메뉴 (menubox.txt) paste명령어를 통해 좌우 통합해서 출력하도록 함

3. Calendar

날짜 계산

```c
// 오늘 날짜 구하기
void get_today(int *year, int *month, int *day) {
        time_t now = time(NULL);
        struct tm *local = localtime(&now);
        *year = local->tm_year + 1900;
        *month = local->tm_mon + 1;
        *day = local->tm_mday;
}

// 윤년 판별
int is_leap_year(int year) {
        return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}

// 월별 일수 계산
int days_in_month(int year, int month) {
        int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        if (month == 2 && is_leap_year(year)) return 29;
        return days[month - 1];
}

// 요일 계산 (0: 일요일, 6: 토요일)
int get_day_of_week(int year, int month, int day) {
        struct tm time_struct = {0};
        time_struct.tm_year = year - 1900;
        time_struct.tm_mon = month - 1;
        time_struct.tm_mday = day;
        mktime(&time_struct);
        return time_struct.tm_wday;
}
```

현재 날짜 가져오는 함수 -> get_today

tm_year은 1900년부터의 경과년수이므로 1900을 더해 실제 연도 계산

tm_mon은 0~11로 표시되므로 1을 더해 실제 월 계산

윤년 계산 함수 -> is_leap_year

(4로 나누어지고 100으로 나누어지지 않거나) OR (400으로 나누어 떨어지면)

월별 일수 지정 (1~12월)

요일 계산 함수 -> get_day_of_week

1900년 1월 1일 월요일을 기준으로 경과된 년수, 개월, 일수 정보로

(기준 요일 + 경과 일수) % 7 = 요일 (0:일요일 ~ 6:토요일)

3. Calendar

일정 관리 기능

```
#define SCHEDULE_FILE "schedules.txt"
```

1) 일정 추가

```
// 일정 추가
void add_schedule(int year, int month, int day, const char* title) {
    FILE *file = fopen(SCHEDULE_FILE, "a");
    if (!file) {
        printf("파일을 열 수 없습니다.\n");
        return;
    }

    fprintf(file, "%d,%d,%d,%s\n", year, month, day, title);
    fclose(file);

    printf("일정이 추가되었습니다: %d년 %d월 %d일 - %s\n", year, month, day, title);
}
```

"a" -> append 모드
파일 끝에 내용 추가, 파일이 없으면 파일 생성

년, 월, 일, 제목 형식으로 저장 후 파일 닫기

3. Calendar

일정 관리 기능

```c
// 월별 일정 보기
void show_month_schedules(int year, int month) {
    FILE *file = fopen(SCHEDULE_FILE, "r");
    if (!file) {
        printf("이번 달에는 일정이 없습니다.\n");
        return;
    }

    Schedule schedules[MAX_SCHEDULES];
    char line[200];
    int count = 0;

    // 해당 월의 모든 일정 읽기
    while (fgets(line, sizeof(line), file) && count < MAX_SCHEDULES) {
        int s_year, s_month, s_day;
        char title[50];

        if (sscanf(line, "%d,%d,%d,%49[^\r\n]", &s_year, &s_month, &s_day, title) == 4) {
            if (s_year == year && s_month == month) {
                schedules[count].year = s_year;
                schedules[count].month = s_month;
                schedules[count].day = s_day;
                my_safe_strcpy(schedules[count].title, title, 50);
                count++;
            }
        }
    }
    fclose(file);

    if (count == 0) {
        printf("이번 달에는 일정이 없습니다.\n");
        return;
    }

    // 날짜순으로 정렬
    qsort(schedules, count, sizeof(Schedule), compare_schedules);

    printf("\n=== %d년 %d월 일정 목록 ===\n", year, month);
    for (int i = 0; i < count; i++) {
        printf("%2d일 : %s\n", schedules[i].day, schedules[i].title);
```

2)월별 일정 보기

"r" -> 읽기 전용으로 파일 열기

fgets()로 파일 한줄씩 읽기

sscanf()로 년, 월, 일, 제목 추출하여 년, 월이 같은 일정 찾기

아래의 일정 비교함수를 이용하여 qsort()로 날짜순 정렬

정렬된 월별 일정 출력

```c
// 일정 비교 함수
int compare_schedules(const void *a, const void *b) {
    Schedule *sched_a = (Schedule *)a;
    Schedule *sched_b = (Schedule *)b;

    if (sched_a->year != sched_b->year) {
        return sched_a->year - sched_b->year;
    }
    if (sched_a->month != sched_b->month) {
        return sched_a->month - sched_b->month;
    }
    return sched_a->day - sched_b->day;
}
```

3. Calendar

일정 관리 기능

```c
// 특정 날짜 일정 보기
void show_day_schedules(int year, int month, int day) {
    FILE *file = fopen(SCHEDULE_FILE, "r");
    if (!file) {
        printf("이 날에는 일정이 없습니다.\n");
        return;
    }

    char line[200];
    char schedules[10][50];
    int count = 0;

    // 해당 날짜의 모든 일정 수집
    while (fgets(line, sizeof(line), file) && count < 10) {
        int s_year, s_month, s_day;
        char title[50];

        if (sscanf(line, "%d,%d,%d,%49[^\r\n]", &s_year, &s_month, &s_day, title) == 4) {
            if (s_year == year && s_month == month && s_day == day) {
                my_safe_strcpy(schedules[count], title, 50);
                count++;
            }
        }
    }
    fclose(file);

    if (count == 0) {
        printf("이 날에는 일정이 없습니다.\n");
        return;
    }
    // 제목순으로 정렬 (간단한 버블 정렬)
    for (int i = 0; i < count - 1; i++) {
        for (int j = i + 1; j < count; j++) {
            if (my_strcmp(schedules[i], schedules[j]) > 0) {
                char temp[50];
                my_strcpy(temp, schedules[i]);
                my_strcpy(schedules[i], schedules[j]);
                my_strcpy(schedules[j], temp);
            }
        }
    }

    printf("\n=== %d년 %d월 %d일 일정 ===\n", year, month, day);
    for (int i = 0; i < count; i++) {
        printf("- %s\n", schedules[i]);
    }
}
```

3)특정 날짜 일정 보기

읽기 모드로 파일 열기, 파일 없으면 종료

년, 월, 일 세가지 조건이 모두 일치하는 일정의 제목만 배열에 저장

그 날에 일정이 없으면 “이 날에는 일정이 없습니다.” 출력

버블 정렬을 이용하여 제목을 사전순으로 정렬

정렬된 일정 출력

3. Calendar


```c
// 날짜별 모든 일정 삭제
void delete_all_schedules_on_date(int year, int month, int day) {
    FILE *file = fopen(SCHEDULE_FILE, "r");
    if (!file) {
        printf("%d년 %d월 %d일에는 일정이 없습니다.\n", year, month, day);
        return;
    }

    char line[200];
    char schedules[10][50];
    int count = 0;

    // 삭제될 일정들을 먼저 수집
    while (fgets(line, sizeof(line), file) && count < 10) {
        int s_year, s_month, s_day;
        char title[50];

        if (sscanf(line, "%d,%d,%d,%49[^\r\n]", &s_year, &s_month, &s_day, title) == 4) {
            if (s_year == year && s_month == month && s_day == day) {
                my_safe_strcpy(schedules[count], title, 50);
                count++;
            }
        }
    }
    fclose(file);

    if (count == 0) {
        printf("%d년 %d월 %d일에는 일정이 없습니다.\n", year, month, day);
        return;
    }

    // 삭제될 일정 목록 보여주기
    printf("\n=== 삭제될 일정 목록 ===\n");
    for (int i = 0; i < count; i++) {
        printf("- %s\n", schedules[i]);
    }
    // 실제 삭제 수행
    file = fopen(SCHEDULE_FILE, "r");
    FILE *temp = fopen("temp.txt", "w");
    if (!file || !temp) {
        printf("파일 처리 중 오류가 발생했습니다.\n");
        return;
    }

    int deleted_count = 0;
    while (fgets(line, sizeof(line), file)) {
        int s_year, s_month, s_day;
        char s_title[50];

        if (sscanf(line, "%d,%d,%d,%49[^\r\n]", &s_year, &s_month, &s_day, s_title) == 4) {
            if (!(s_year == year && s_month == month && s_day == day)) {
                fprintf(temp, "%s", line);
            } else {
                deleted_count++;
            }
        } else {
            fprintf(temp, "%s", line);
        }
    }

    fclose(file);
    fclose(temp);

    if (deleted_count > 0) {
        remove(SCHEDULE_FILE);
        rename("temp.txt", SCHEDULE_FILE);
        printf("%d개의 일정이 삭제되었습니다.\n", deleted_count);
    } else {
        remove("temp.txt");
        printf("삭제할 일정이 없습니다.\n");
```

4)일정 삭제

삭제하려는 년, 월, 일 정보가 일치하는 일정의 제목을 임시 저장

일정이 없는 경우 “일정이 없습니다.” 출력

일정이 있는 경우 일정 목록 보여주기

원본파일을 읽기 모드로 열기, 임시 파일(temp.txt)를 쓰기 모드로 열기 - 열기 실패시 오류 메시지

삭제하려는 일정의 년, 월, 일 정보와 일치하지 않으면
삭제 대상이 아니므로 임시파일에 복사
삭제 대상인 경우 임시파일에 복사하지 않고 deleted_count 증가

deleted_count > 0인 경우 원본 파일(schedules.txt) 삭제 후 임시파일 이름을 원본 파일으로 변경
조건 만족하지 않으면 임시 파일만 삭제

3. Calendar


```
# 특정 달로 이동
go_to_month() {
        echo
        echo -e "${CYAN}${BOLD}=== 특정 달로 이동 ===${RESET}"
        echo

        echo -n "년도 (현재: $CURRENT_YEAR): "
        read year
        if [ -z "$year" ]; then
                year=$CURRENT_YEAR
        fi

        echo -n "월 (현재: $CURRENT_MONTH): "
        read month
        if [ -z "$month" ]; then
                month=$CURRENT_MONTH
        fi

        # 유효성 검사
        if ! is_number "$year" || ! is_number "$month"; then
                echo -e "${RED}잘못된 날짜 형식입니다. 숫자만 입력하세요.${RESET}"
                echo -e "${YELLOW}Enter를 눌러 계속...${RESET}"
                read
                return
        fi

        if [ $month -lt 1 ] || [ $month -gt 12 ]; then
                echo -e "${RED}월은 1-12 사이의 값이어야 합니다.${RESET}"
                echo -e "${YELLOW}Enter를 눌러 계속...${RESET}"
                read
                return
        fi

        CURRENT_YEAR=$year
        CURRENT_MONTH=$month
}
```

5) 특정 달로 이동

현재 년도 참고 정보로 표시
년도 입력에서 Enter만 누르면 현재 년도 사용 or 원하는 년도 입력

현재 월 참고 정보로 표시
월 입력에서 Enter만 누르면 현재 월 사용 or 원하는 월 입력

년도와 월 모두 입력값이 숫자인지 검증하고 숫자가 아니면 “잘못된 입력” 메시지
월 범위 (1월 ~ 12월)인지 검증 후 범위를 벗어나면 “1~12 사이의 값 입력” 메시지

조건 충족 시 메인루프에서 특정 달로 이동

3. Calendar

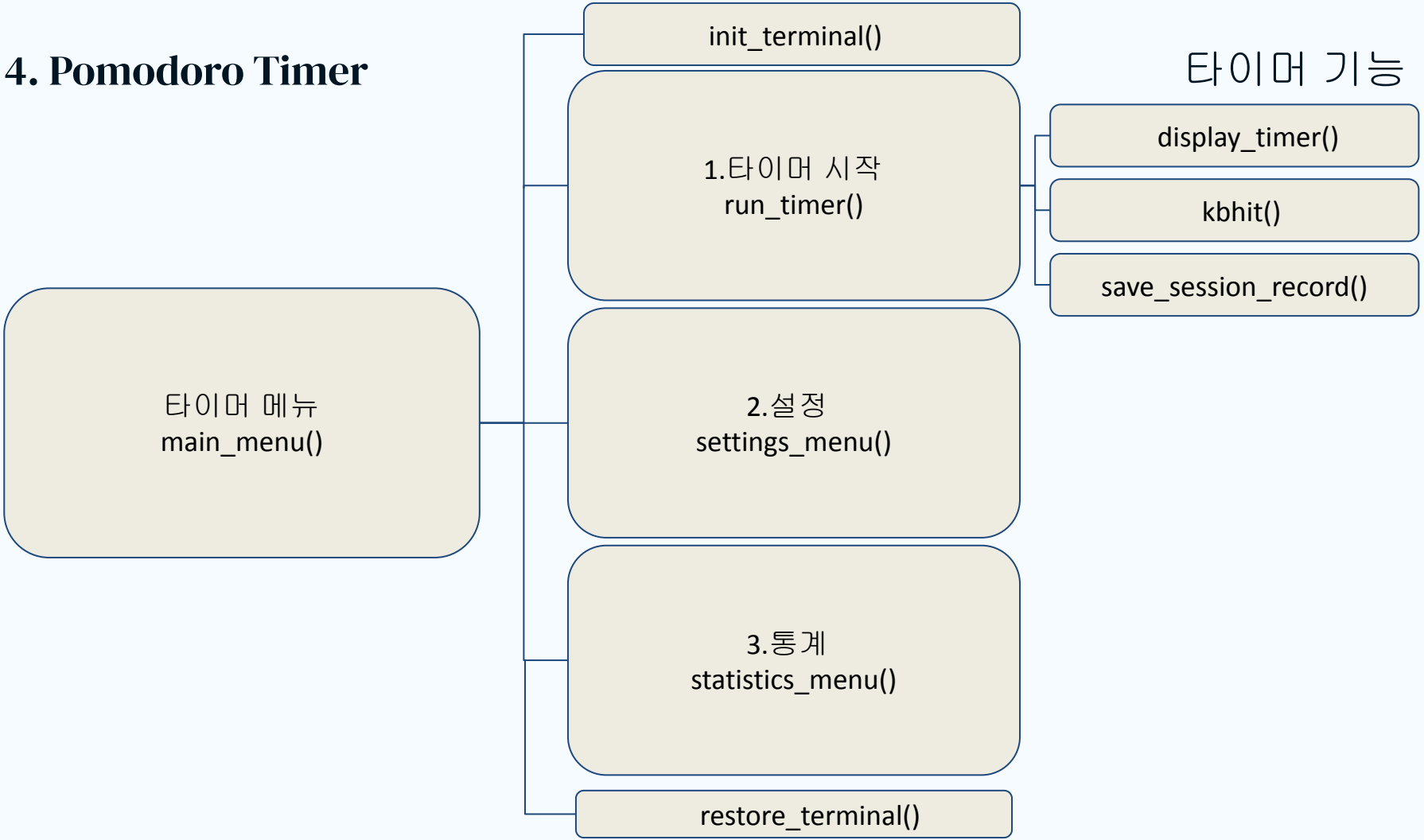
메뉴

```
# 메인 루프
while true; do
        show_main_screen
        read choice

        case $choice in
                1) add_schedule ;;
                2) show_all_schedules ;;
                3) show_day_schedules ;;
                4) delete_schedules ;;
                5)
                        if [ $CURRENT_MONTH -eq 1 ]; then
                                CURRENT_YEAR=$((CURRENT_YEAR - 1))
                                CURRENT_MONTH=12
                        else
                                CURRENT_MONTH=$((CURRENT_MONTH - 1))
                        fi
                        ;;
                6)
                        if [ $CURRENT_MONTH -eq 12 ]; then
                                CURRENT_YEAR=$((CURRENT_YEAR + 1))
                                CURRENT_MONTH=1
                        else
                                CURRENT_MONTH=$((CURRENT_MONTH + 1))
                        fi
                        ;;
                7) go_to_month ;;
                8) run_terminal_chat ;;
                9) run_pomodoro_timer ;;
                0) exit 0 ;;
                *)
                        echo -e "${RED}잘못된 선택입니다. 0-9 사이의 숫자를 입력하세요.${RESET}"
                        echo -e "${YELLOW}Enter를 눌러 계속...${RESET}"
                        read
                        ;;
        esac
done
```

무한루프로 exit하기 전까지 캘린더와 메뉴화면 출력하고 선택 입력 받기

1)일정 추가
2)이번 달 모든 일정 보기
3)특정 날짜 일정 보기
4)일정 삭제 (날짜별)
5)이전 달
    1월이면 년도 -1 & 12월 설정, 1월이 아니면 월만 -1
6)다음 달
    12월이면 년도 +1 & 1월 설정, 12월이 아니면 월만 +1
7)특정 달로 이동
8)터미널 채팅
9)뽀모도로 타이머
0)종료
*)(1~0)의 입력 외의 경우 잘못된 입력 처리

4. Pomodoro Timer



4. Pomodoro Timer

타이머 기능

```
void main_menu() {
    int completed_sessions = 0;

    while (1) {
        printf("\033[2J\033[H");  // 화면 지우기
        printf("=== 뽀모도로 타이머 ===\n\n");
        printf("1. 타이머 시작\n");
        printf("2. 설정\n");
        printf("3. 통계\n");
        printf("0. 종료\n\n");
        printf("선택: ");

        int choice;
        if (scanf("%d", &choice) != 1) {
            while (getchar() != '\n');
            continue;
        }
        while (getchar() != '\n');

        switch (choice) {
            case 1:
                run_timer(work_time, "작업");
                completed_sessions++;

                if (completed_sessions % sessions_before_long_break == 0) {
                    run_timer(long_break, "긴 휴식");
                } else {
                    run_timer(short_break, "짧은 휴식");
                }
                break;
            case 2:
                settings_menu();
                break;
            case 3:
                statistics_menu();
                break;
            case 0:
                return;
        }
    }
```

뽀모도로 타이머 main code

처음 실행시 printf("\033[2J\033[H"); 코드를 통해 터미널 화면의 모든 내용을 지우고 커서를 화면 맨위 왼쪽으로 이동 시킴 및 화면 출력

if (scanf("%d", &choice) != 1) 코드를 통해 함수로 입력 받은 값을 반환하고 abc같은 글자 입력시 수자 읽기를 실패하고 0을 반환

화면 출력 후 각 case0~3 까지
1 실제 뽀모도로 작업이 일어나며
2 setting_menu();;를 호출 하여 설정 화면으로 이동
3 statistics_menu ()로 통계화면으로 이동
0 main_menu함수 종료 후 프로그램 종료

4. Pomodoro Timer

```c
// 타이머 실행 함수
void run_timer(int duration, const char* phase) {
    int total_seconds = duration * 60;
    int remaining_seconds = total_seconds;
    int paused = 0;
    time_t start_time = time(NULL);
    time_t pause_start = 0;
    int total_pause_time = 0;

    // 타이머 실행 중에는 ECHO를 끄고
    struct termios timer_termios;
    tcgetattr(STDIN_FILENO, &timer_termios);
    timer_termios.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &timer_termios);

    while (remaining_seconds > 0) {
        if (!paused) {
            time_t current_time = time(NULL);
            int elapsed = current_time - start_time - total_pause_time;
            remaining_seconds = total_seconds - elapsed;

            if (remaining_seconds < 0) remaining_seconds = 0;

            int minutes = remaining_seconds / 60;
            int seconds = remaining_seconds % 60;

            display_timer(minutes, seconds, total_seconds, phase);
        }

        // 키 입력 확인
        if (kbhit()) {
            char key = getchar();
            if (key == 'p' || key == 'P') {
                if (!paused) {
                    paused = 1;
                    pause_start = time(NULL);
                    printf("\n일시정지 중... (p를 눌러 재개)\n");
                } else {
                    paused = 0;
                    total_pause_time += time(NULL) - pause_start;
                }
            } else if (key == 'q' || key == 'Q') {
                printf("\n타이머를 종료합니다.\n");
                // 타이머 종료 시 원래 터미널 설정으로 복원
                tcsetattr(STDIN_FILENO, TCSANOW, &orig_termios);
                return;
            }
        }

        usleep(100000);  // 0.1초 대기
    }
```

타이머 기능

뽀모도로 타이머 run_timer

함수 시작 되면 total_seconds 에서 초기 25분으로 세팅 된 것을 초단위로 변환시킴

start_time = time(NULL) 을 통해 현재 시간을 초단위의 긴시간으로 반환하여 타이머가 시작된 정확한 시점을 기록함

total_pause_time변수를 통해 사용자가 타이머를 일시정지한 시간의 총합을 저장

int elapsed = current_time - start_time - total_pause_time; 함수를 통해 단순히 1초씩 빼는 방식이 아니라 (현재시간-시작시간 -총 일시정지 시간)을 계산하여 실제로 흘러간 순수시간을 구함

kbhit()를 통해 타이머의 일시정지 p와 타이머의 종료 q를 동작함

타이머가 0초가 되어 루프가 정상적으로 끝나면 q를 눌렀을때와 마찬가지로 터미널 설정을 원래대로 복원 후 save_session_record함수를 호출해 작업 또는 휴식 세션이 완료된것을 로그 파일에 기록 함

4. Pomodoro Timer

타이머 기능

```c
// 설정 메뉴 함수
void settings_menu() {
    while (1) {
        printf("\033[2J\033[H");  // 화면 지우기
        printf("=== 설정 ===\n\n");
        printf("1. 작업 시간 설정 (현재: %d분)\n", work_time);
        printf("2. 짧은 휴식 시간 설정 (현재: %d분)\n", short_break);
        printf("3. 긴 휴식 시간 설정 (현재: %d분)\n", long_break);
        printf("4. 작업 이름 설정 (현재: %s)\n", strlen(current_task) > 0 ? current_task : "없음");
        printf("5. 기본값으로 복원\n");
        printf("0. 돌아가기\n\n");
        printf("선택: ");

        int choice;
        if (scanf("%d", &choice) != 1) {
            while (getchar() != '\n');
            continue;
        }
        while (getchar() != '\n');

        switch (choice) {
            case 1:
                printf("작업 시간(분): ");
                if (scanf("%d", &work_time) != 1) {
                    while (getchar() != '\n');
                    continue;
                }
                while (getchar() != '\n');
                break;
            case 2:
                printf("짧은 휴식 시간(분): ");
                if (scanf("%d", &short_break) != 1) {
                    while (getchar() != '\n');
                    continue;
                }
                while (getchar() != '\n');
                break;
            case 3:
                printf("긴 휴식 시간(분): ");
                if (scanf("%d", &long_break) != 1) {
                    while (getchar() != '\n');
                    continue;
                }
                while (getchar() != '\n');
                break;
            case 4:
                printf("작업 이름: ");
                if (fgets(current_task, sizeof(current_task), stdin) == NULL) {
                    continue;
                }
                current_task[strcspn(current_task, "\n")] = 0;
                break;
            case 5:
                work_time = WORK_TIME;
                short_break = SHORT_BREAK;
                long_break = LONG_BREAK;
                current_task[0] = '\0';
                printf("기본값으로 복원되었습니다.\n");
                sleep(1);
                break;
            case 0:
                return;
```

settings_menu

뽀모도로 타이머의 작업시간 설정, 짧은 휴식 시간 설정, 긴휴식 시간 설정, 작업이름 설정, 기본값으로 복원 하는 기능을 구현함

```
=== 설정 ===

1. 작업 시간 설정 (현재: 25분)
2. 짧은 휴식 시간 설정 (현재: 5분)
3. 긴 휴식 시간 설정 (현재: 15분)
4. 작업 이름 설정 (현재: 없음)
5. 기본값으로 복원
0. 돌아가기
```

4. Pomodoro Timer

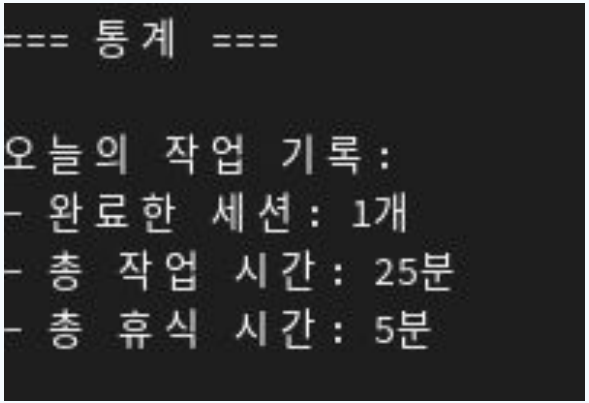
타이머 기능

```c
// 통계 메뉴 함수
void statistics_menu() {
    printf("\033[2J\033[H");  // 화면 지우기
    printf("=== 통계 ===\n\n");

    time_t now = time(NULL);
    struct tm* t = localtime(&now);
    char filename[FILENAME_BUFFER_SIZE];
    snprintf(filename, FILENAME_BUFFER_SIZE, "pomodoro_log_%04d%02d%02d.txt", t->tm_year + 1900, t->tm_mon + 1, t->tm_mday

    FILE* file = fopen(filename, "r");
    if (file) {
        char line[256];
        int total_work_time = 0;
        int total_breaks = 0;
        int sessions = 0;

        while (fgets(line, sizeof(line), file)) {
            if (strstr(line, "작업") || strstr(line, "Work")) {
                total_work_time += work_time;
                sessions++;
            } else if (strstr(line, "휴식") || strstr(line, "Break")) {
                total_breaks += (strstr(line, "긴") || strstr(line, "Long")) ? long_break : short_break;
            }
        }

        printf("오늘의 작업 기록:\n");
        printf("- 완료한 세션: %d개\n", sessions);
        printf("- 총 작업 시간: %d분\n", total_work_time);
        printf("- 총 휴식 시간: %d분\n", total_breaks);

        fclose(file);
    } else {
        printf("오늘의 작업 기록이 없습니다.\n");
    }

    printf("\nEnter를 눌러 돌아가기...");
    getchar();
}
```

뽀모도로 타이머 statistics_menu

하루동안 공부하거나 일한 내용을 요약해서
보여주는 기능

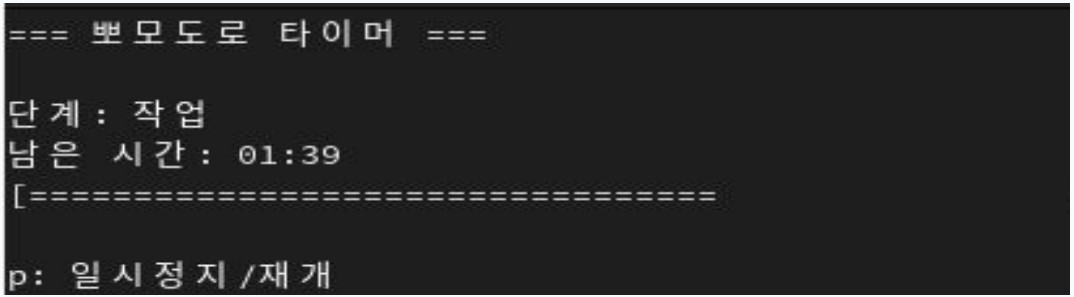
오늘 날짜의 기록 파일을 만들고
그파일이 있으면 내용을 분석해 총 몇번 집중했고
총 몇분 공부했는지 계산해서 화면에 보여줌



4. Pomodoro Timer

타이머 기능

```c
// 타이머 표시 함수
void display_timer(int minutes, int seconds, int total_seconds, const char* phase) {
    printf("\033[2J\033[H");  // 화면 지우기
    printf("=== 뽀모도로 타이머 ===\n\n");

    if (strlen(current_task) > 0) {
        printf("현재 작업: %s\n\n", current_task);
    }

    printf("단계: %s\n", phase);
    printf("남은 시간: %02d:%02d\n", minutes, seconds);

    // 진행률 표시
    display_progress(total_seconds - (minutes * 60 + seconds), total_seconds, 50);

    printf("\n\n");
    printf("p: 일시정지/재개\n");
    printf("q: 종료\n");
    fflush(stdout);
}
```



minutes:남은 분
seconds:남은 초
total_seconds: 전체 시간(초)

화면을 지우고 남은 시간을 출력하는 과정을 반복하여 사용자가 실시간으로 남은 시간과 진행률을 볼 수 있는 구조

display_progress()로 작업률 바를 실시간으로 출력함

4. Pomodoro Timer

타이머 기능

```c
// 진행률 표시 함수
void display_progress(int current, int total, int width) {
    float progress = (float)current / total;
    int filled = (int)(progress * width);

    printf("\r[");
    for (int i = 0; i < width; i++) {
        if (i < filled) printf("=");
        else printf(" ");
    }
    printf("] %d%%", (int)(progress * 100));
    fflush(stdout);
}
```

current: 현재 작업한 시간(초)
progress: 진행률
filled: 진행한 양(칸)

전체 시간에 대해 진행률을 계산해
반복문으로 진행률 바를 출력하는 구조

4. Pomodoro Timer

타이머 기능

```c
// 세션 기록 저장 함수
void save_session_record(const char* phase, int duration) {
    time_t now = time(NULL);
    struct tm* t = localtime(&now);
    char filename[32];
    sprintf(filename, "pomodoro_log_%04d%02d%02d.txt", t->tm_year + 1900, t->tm_mon + 1, t->tm_mday);

    FILE* file = fopen(filename, "a");
    if (file) {
        char task_info[120] = "";
        if (strlen(current_task) > 0) {
            sprintf(task_info, " (작업: %s)", current_task);
        }
        fprintf(file, "[%02d:%02d:%02d] %s 완료%s\n",
                t->tm_hour, t->tm_min, t->tm_sec, phase, task_info);
        fclose(file);
    }
}
```

작업,휴식 등이 끝나면

파일 이름을
pomodoro_log_20250619.txt의
형식으로 만들고

파일에
[19:40:02] 작업 완료 (작업: ppt)의
형식으로 기록한다.

통계 기능에서 오늘 어떤 작업을
했는지 조회할 때 쓰인다

4. Pomodoro Timer

타이머 기능

```c
// 터미널 설정 초기화
void init_terminal() {
    tcgetattr(STDIN_FILENO, &orig_termios);
    struct termios new_termios = orig_termios;
    new_termios.c_lflag &= ~ICANON;  // ECHO는 유지
    tcsetattr(STDIN_FILENO, TCSANOW, &new_termios);
}

// 터미널 설정 복원
void restore_terminal() {
    tcsetattr(STDIN_FILENO, TCSANOW, &orig_termios);
}
```

```
일시정지 중... (p를 눌러 재개)
```

사용자의 입력을 문자 단위로 한 글자씩 입력 받는 모드

메뉴에서 한 글자씩 입력 받기 위해 사용됨

tcgetattr(STDIN_FILENO,&orig_termios):원래 터미널 상태를 나중에 복원 할 수 있도록 orig_termios에 저장

new_termios.c_1flag &=~ICANON: 캐노니컬 모드를 꺼서 한 글자씩 입력 받게한다.

tcsetattr(STDIN_FILENO,TCSANOW,&new_termios):현재 터미널을 새로 설정한 new_termios로 바꿈

4. Pomodoro Timer

타이머 기능

```c
// kbhit 함수 구현
int kbhit() {
    struct termios oldt, newt;
    int ch;
    int oldf;

    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

    ch = getchar();

    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    fcntl(STDIN_FILENO, F_SETFL, oldf);

    if (ch != EOF) {
        ungetc(ch, stdin);
        return 1;
    }

    return 0;
}
```

p 또는 q를 누르면 타이머가 정지하는 리얼타임 시스템 구현을 위해 사용 됨

fcntl():파일 디스크립터의 동작을 제어하는시스템 호출

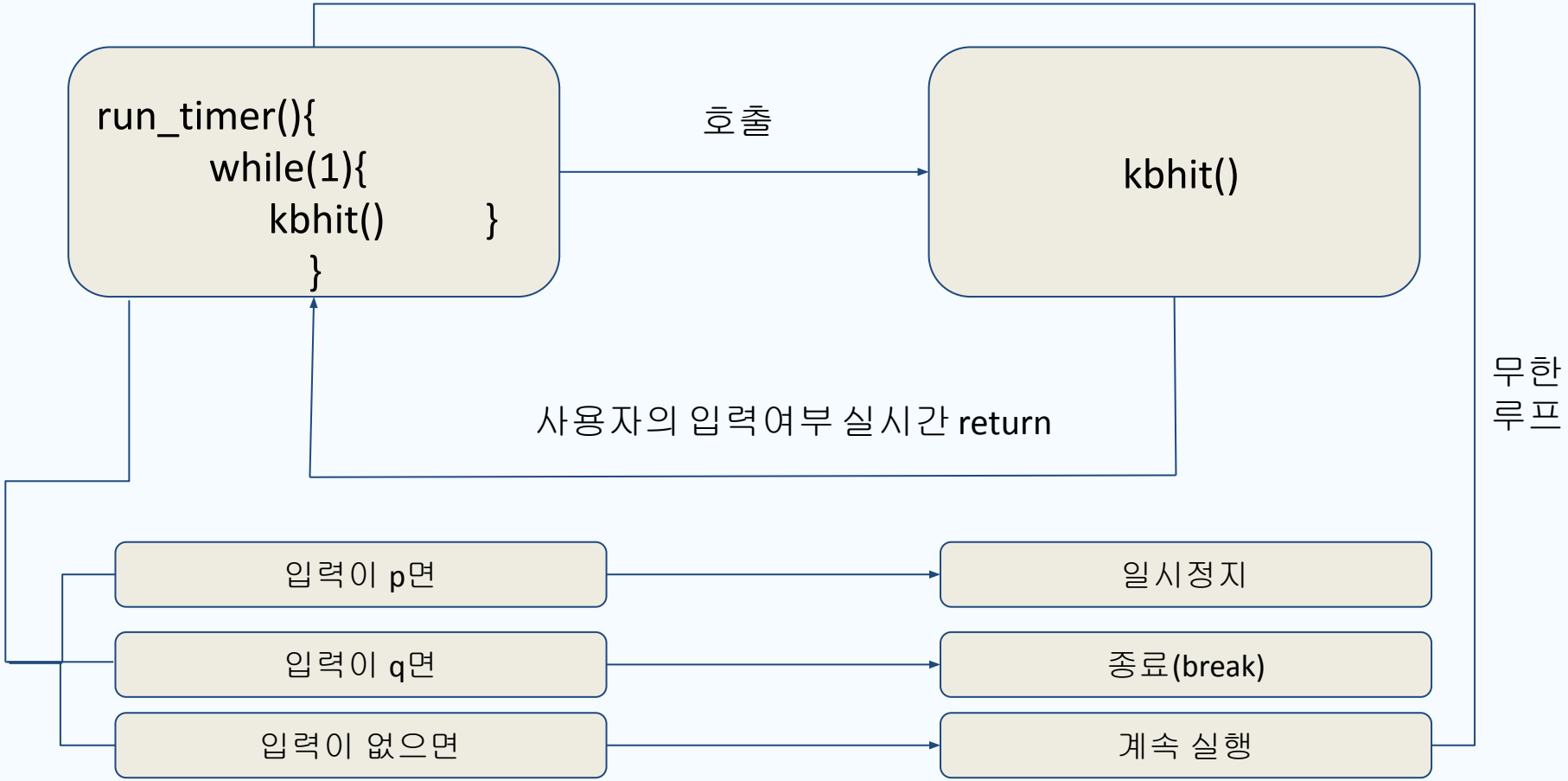
oldf = fcntl(STDIN_FILENO,F_GETFL,0):현재 키보드 입력의 설정을 가져와서 저장

원래 터미널: 입력을 계속 기다림

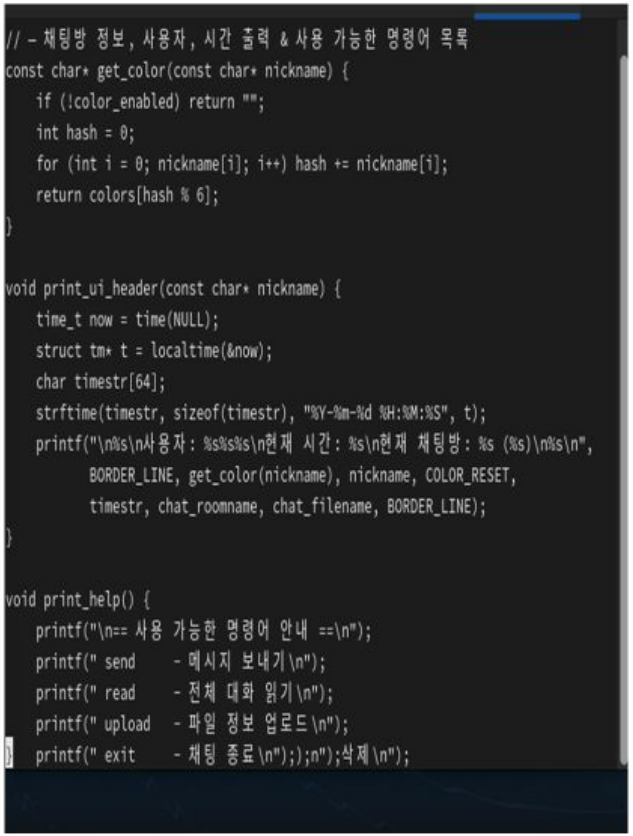
fcntl(STDIN_FILENO,F_SETFL,oldf|O_NONBLOCK):
터미널 입력 대기를 하지 않도록 설정
입력이 있으면 해당 문자를 즉시반환하고
입력이 없으면 EOF를 반환 하도록 설정

4. Pomodoro Timer

타이머 기능



5. Lightweight chat



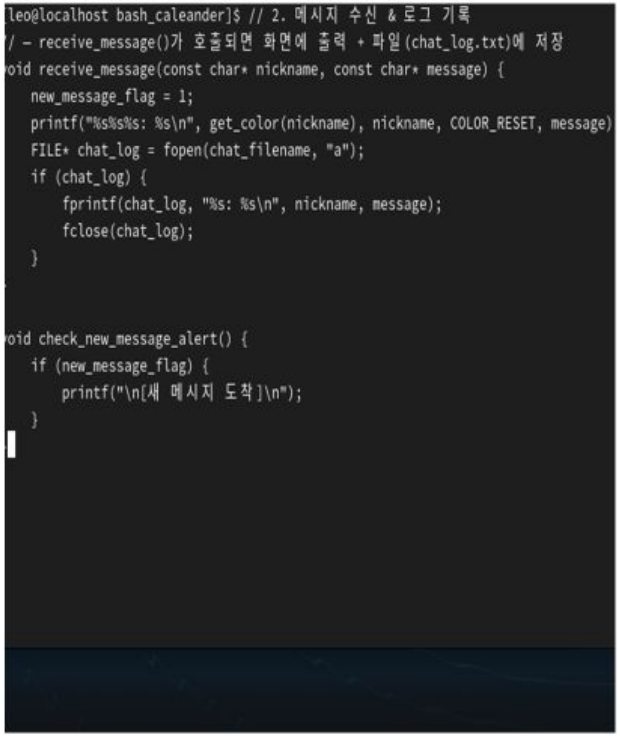
```
// - 채팅방 정보, 사용자, 시간 출력 & 사용 가능한 명령어 목록
const char* get_color(const char* nickname) {
    if (!color_enabled) return "";
    int hash = 0;
    for (int i = 0; nickname[i]; i++) hash += nickname[i];
    return colors[hash % 6];
}

void print_ui_header(const char* nickname) {
    time_t now = time(NULL);
    struct tm* t = localtime(&now);
    char timestr[64];
    strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", t);
    printf("\n%s\n사용자 : %s%s%s\n현재 시간 : %s\n현재 채팅방 : %s (%s)\n%s\n",
           BORDER_LINE, get_color(nickname), nickname, COLOR_RESET,
           timestr, chat_roomname, chat_filename, BORDER_LINE);
}

void print_help() {
    printf("\n== 사용 가능한 명령어 안내 ==\n");
    printf(" send     - 메시지 보내기\n");
    printf(" read     - 전체 대화 읽기\n");
    printf(" upload   - 파일 정보 업로드\n");
    printf(" exit     - 채팅 종료\n"););n");삭제\n");
}
```

•print_ui_header()가 테두리와 함께
•현재 시간, 채팅방 이름, 로그 파일명을 출력해
•사용자 닉네임을 컬러로 보임

•get_color()로 닉네임별 색상을 정해서
•strftme()으로 ‘YYYY-MM-DD HH:MM:SS’ 포맷 시간 표시

•print_help()가 사용 가능한 명령어 전체 목록(send, read, upload, search, tagsearch, tags, delete, change, help, exit)을 한눈에 보임

5. Lightweight chat



```
leo@localhost bash_caleander]$ // 2. 메시지 수신 & 로그 기록
/ – receive_message()가 호출되면 화면에 출력 + 파일(chat_log.txt)에 저장
oid receive_message(const char* nickname, const char* message) {
    new_message_flag = 1;
    printf("%s%s%s: %s\n", get_color(nickname), nickname, COLOR_RESET, message);
    FILE* chat_log = fopen(chat_filename, "a");
    if (chat_log) {
        fprintf(chat_log, "%s: %s\n", nickname, message);
        fclose(chat_log);
    }


oid check_new_message_alert() {
    if (new_message_flag) {
        printf("\n[새 메시지 도착]\n");
    }
```

- receive_message()가 호출되면
- new_message_flag=1 세팅해서 새 메시지 상태로 표시
- printf()로 닉네임(컬러)과 메시지를 즉시 화면에 출력
- chat_log.txt에 nickname: message 형식으로 한 줄씩 저장
- check_new_message_alert()가
- new_message_flag 확인 후
- “[새 메시지 도착]” 문구를 출력해 사용자에게 전달

5. Lightweight chat

```
[leo@localhost bash_caleander]$  3. 채팅 기록 읽기 (read 명령 처리)
// - chat_log.txt를 열어 전체 대화 출력
void read_messages() {
    new_message_flag = 0;
    printf("\n[채팅 기록]\n");
    FILE* chat_log = fopen(chat_filename, "r");
    if (chat_log) {
        char line[MAX_LINE];
        while (fgets(line, sizeof(line), chat_log)) {
            printf("%s", line);
        }
        fclose(chat_log);
    } else {
        printf("(채팅 기록이 없습니다)\n");
    }
}
```

- read_messages()가 실행되면
  - •new_message_flag=0으로 초기화해서 알림 해제
  - •fopen("chat_log.txt","r")로 로그 파일 열기
  - •while(fgets(...))로 파일 끝까지 한 줄씩 읽어서 printf()로 출력
  - •파일 없거나 열기 실패 시 “(채팅 기록이 없습니다)” 표시
  - •fclose()로 파일을 닫고 리소스 정리

5. Lightweight chat

```
leo@localhost bash_caleander]$ // 4. 파일 업로드 (upload 명령 처리)
/ – 파일명, 설명, 태그를 입력 받아 files.txt에 메타데이터 저장
oid upload_file(const char* nickname) {
   char filename[64], description[128], tags[128];
   printf("\n[파일 업로드]\n파일명: "); scanf("%s", filename); getchar();
   printf("설명: "); fgets(description, sizeof(description), stdin);
   description[strcspn(description, "\n")] = 0;
   printf("태그 (쉼표로 구분): "); fgets(tags, sizeof(tags), stdin);
   tags[strcspn(tags, "\n")] = 0;

   FILE* file_meta = fopen("files.txt", "a");
   if (file_meta) {
       fprintf(file_meta, "%s - %s [%s] (%s)\n",
               filename, description, tags, nickname);
       fclose(file_meta);
       printf("업로드 완료\n");
   } else {
       printf("업로드 실패\n");
   }
```

- upload_file()이 호출되면
  - “파일명: ”, “설명: ”, “태그(쉼표구분): ” 순서로 사용자 입력 받고
  - files.txt를 append 모드("a")로 열어
  - fprintf(...,"%s - %s [%s] (%s)\n", filename, description, tags, nickname) 한 줄 기록
  - 업로드 성공 시 “업로드 완료” 출력, 실패 시 “업로드 실패” 알림
  - 메타데이터를 깔끔하게 저장해서 나중에 검색·삭제 가능

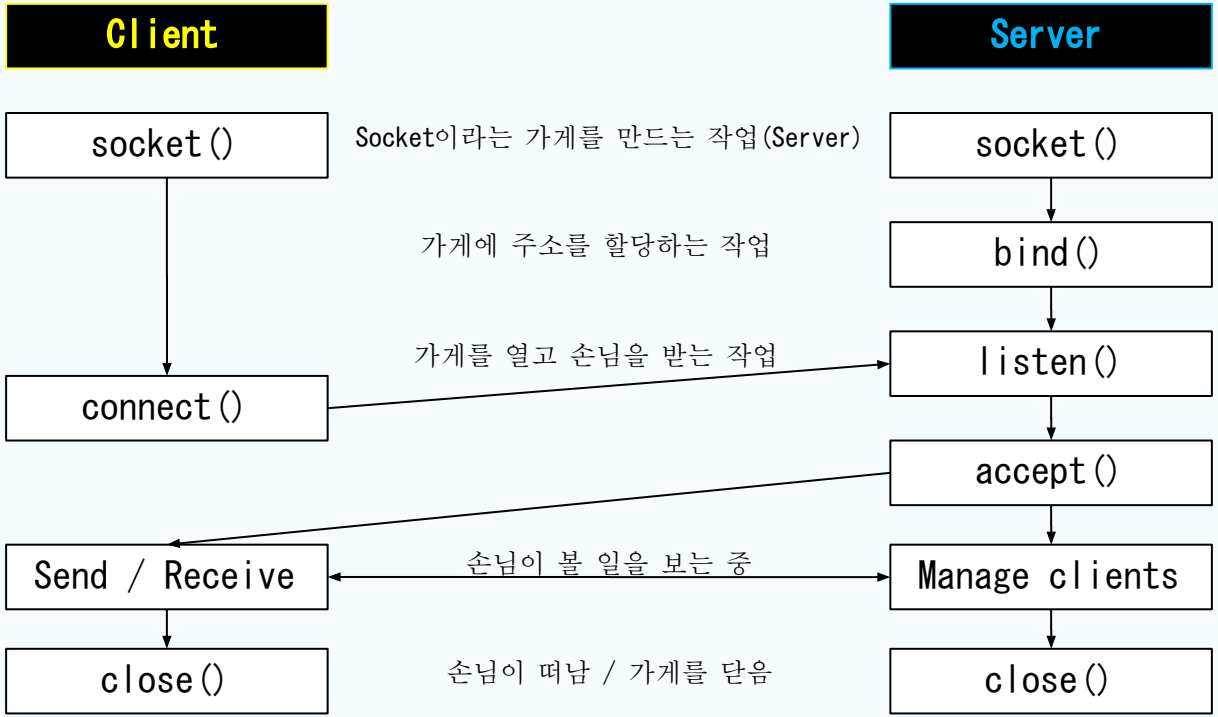
5. Lightweight chat

```
[leo@localhost bash_caleander]$ // 5. 키워드 & 태그 검색 (search, tagsearch 명령 처리)
// - files.txt에서 키워드 혹은 태그를 찾아 출력
void search_file(const char* keyword) {
    FILE* fp = fopen("files.txt", "r");
    if (!fp) { perror("파일 열기 실패"); return; }
    char line[MAX_LINE]; int found = 0;
    printf("\n[검색 결과: '%s']\n", keyword);
    while (fgets(line, sizeof(line), fp)) {
[leo@localhost bash_caleander]$ // 5. 키워드 & 태그 검색 (search, tagsearch 명령 처리)
// - files.txt에서 키워드 혹은 태그를 찾아 출력
void search_file(const char* keyword) {
    FILE* fp = fopen("files.txt", "r");
    if (!fp) { perror("파일 열기 실패"); return; }
    char line[MAX_LINE]; int found = 0;  없습니다)\n");
    printf("\n[검색 결과: '%s']\n", keyword);
    while (fgets(line, sizeof(line), fp)) {
        if (strstr(line, keyword)) {
            printf("%s", line);* tag) {
            found = 1;files.txt", "r");
        )fp) { perror("파일 열기 실패"); return; }
    }har line[MAX_LINE]; int found = 0;
    if (!found) printf("(일치하는 결과가 없습니다)\n");
    fclose(fp);s(line, sizeof(line), fp)) {
}       char* start = strchr(line, '[');
        char* end   = strchr(line, ']');
void tag_search_file(const char* tag) {
    FILE* fp = fopen("files.txt", "r");
    if (!fp) { perror("파일 열기 실패"); return; };
    char line[MAX_LINE]; int found = 0;;
    printf("\n[태그 검색 결과: '%s']\n", tag);
    while (fgets(line, sizeof(line), fp)) {
        char* start = strchr(line, '[');
        char* end   = strchr(line, ']');
        if (start && end) {
            char section[128];
            strncpy(section, start+1, end-start-1);n");
            section[end-start-1] = '\0';
            if (strstr(section, tag)) {
                printf("%s", line);
                found = 1; 치하는 태그가 없습니다)\n");
            }
        }
    }
    if (!found) printf("(일치하는 태그가 없습니다)\n");
    fclose(fp);
}
```

- search_file()
  - fopen("files.txt","r")로 파일 열어
  - while(fgets(...))로 한 줄씩 읽어
  - strstr(line, keyword)로 키워드 존재 여부 검사 후 일치하는 줄만 printf()
  - 결과 없으면 “(일치하는 결과가 없습니다)” 표시
- tag_search_file()
  - 각 줄에서 [와] 사이 문자열을 추출해서
  - strstr(tag_section, tag)로 태그 일치 검사
  - 일치할 때만 출력, 없으면 “(일치하는 태그가 없습니다)” 알림

5. Chatting

채팅 기능



socket(domain, type, protocol)

- 반환값: Socket Descriptor – 운영체제가 부여한 가게의 번호
- Domain: **IPv4**, IPv6, UNIX, IPX, Low-Level Socket Interface Protocol 중 선택(PF_INET)
- Type: **TCP**, UDP, RAW 형태 중 선택(SOCK_STREAM)
- Protocol: **TCP**, UDP Protocol 중 선택(자동: IPPROTO_HOPOPTS 또는 **0**)
- 가게의 운영**(통신) 방식**과 **주소 체계**를 **확립**하는 함수

bind(SD, &address, length)

- 반환값: 오류 여부
- Socket Descriptor: 가게 번호
- Address: 가게의 **논리 주소**가 저장되어 있는 **구조체 Memory 주소**
- Length: 가게의 **논리 주소**가 저장되어 있는 **구조체 크기(Memory 영역)**
- 왜 직접 구조체 변수를 가져오지 않는가?: 논리 주소 구조체처럼 여러 형태를 갖는 변수를 인자로 받는 함수의 **다형성 대응**을 위해 C 언어의 함수는 인자의 형식을 지정해 직접 가져오지 않고 **Memory 위치 및 영역에 기반한 참조**로 인자를 가져오는 경우가 많음
- 가게에 **주소**를 **할당**하는 함수

listen(SD, backlog)

- 반환값: 오류 여부
- Socket Descriptor: 가게 번호
- Backlog: 가게의 **입구 크기** – 가게의 수용 한계를 의미하는 것이 아님
- 가게 **문을 열고 손님**을 **대기**하는 함수

accept(SD, &address, &length)

- 반환값: **연결이 허락된 Client**에 운영체제가 부여하는 **Socket Descriptor**(또는 오류 -1)
- Socket Descriptor: (Server의) 가게 번호
- Address: **Client의 논리 주소**가 저장될 **구조체 Memory 주소**
- *Length: **Client의 논리 주소**가 저장될 **구조체 크기 정보**(가 담긴 변수의 Memory 주소가 인자)
- 수행되면 Backlog에 **손님**의 **방문(연결 요청)**이 있을 때까지 **대기** 상태에 돌입
- 가게에 **들어온 손님**의 **정보**를 **기록**하는 함수

close(SD)

- 반환값: 오류 여부
- Socket Descriptor: 가게 번호
- 손님(Client)은 가게의 **논리 주소 등본**을 **폐기**하고 가게(Server)는 **문을 닫고 정리**하는 함수

Mutex(Mutual Exclusion: 상호 배제)

- 손님이 출입하는 와중에 가게가 이 정보를 관리하다 보면 **아직 비어 있지 않은 자리에 손님을 안내**하거나 **비어 있는 자리가 있음에도 인지하지 못하는 상황**이 발생 – 정보를 관리할 때 사용하는 변수가 **출입을 관리하는 함수들의 전역 변수(공유 자원)**이기 때문
- **Process 내에서 Thread가 복수로 분리(점원이 둘 이상)**되면 이 **함수들이 동시에 실행**될 수 있기 때문에 문제가 발생하는 것
- 이 기록부(공유 자원)를 이미 사용 중인 Thread(점원) 외에는 사용할 수 없도록 통제하는 기법 중 하나
- Acquire 명령으로 Key를 가져간 Thread만 임계 구역에 접근 가능하며, 일을 끝마친 Thread가 Release 명령으로 Key를 반납하는 체계
- C에서는 mutex 구조체를 전역 변수로 선언 후 각각 pthread_mutex_lock(&mutex), pthread_mutex_unlock(&mutex)으로 구현

5. Chatting

채팅 기능

```
pthread_mutex_t mutex;
int clients_num;
int clients[100]= {[0 ... 99]= -1};
FILE* chat_log;
time_t tm;
struct tm current_time;

int space_available() {
    int i;
    for(i= 0; i < 100; i++) {
        if(clients[i] < 0)
            return i;
    }

    return i;
}

void pass_message(char* message, int message_length) {
    pthread_mutex_lock(&mutex);
        int i;
        char ct[24];

        for(i= 0; i < 100; i++) {
            if(clients[i] >= 0 && message_length-1) {
                write(clients[i], message, message_length);

                tm= time(NULL);
                current_time= *localtime(&tm);
                sprintf(ct, "%d-%d-%d %d:%d:%d ", current_time.tm_year+1900, curre
                fputs(ct, chat_log);
                fputs(message, chat_log);

                printf("%d에게 다음 내용 전송: %s\n", clients[i], message);
            }
        }
    pthread_mutex_unlock(&mutex);
}
```

Server

- **전역 변수 목록**
  - mutex: Mutex 구조체
  - clients_num: 접속 중인 Client 총원
  - clients: Client들의 SD 모음
  - chat_log: Chatting 내역 기록을 위한 File의 FD
- **사용자 정의 함수 목록**
  - space_available(): 비어 있는 SD 지정 공간 찾기
  - pass_message(): 수신한 Message를 접속 중인 모든 Client들에게 제공하고 그 내역을 File에 추가(마지막 Client가 접속을 종료하면 내역 File을 저장)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>
#include <time.h>
```

5. Chatting

채팅 기능

```
void* manage_client(void* client_socket) {
    int message_length;
    char message[1024];

    while(message_length= read(*(int*)client_socket, message, sizeof(message))) {
        if(!(message_length-1)) break;

        pass_message(message, message_length);
        printf("길이: %d, 연결 유지 중\n", message_length);
    }

    printf("%d의 연결 해제 발생\n", *(int*)client_socket);

    pthread_mutex_lock(&mutex);
        int i;
        for(i= 0; i < 100; i++) {
            if(*(int*)client_socket == clients[i]) {
                clients[i]= -1;
                break;
            }
        }
        --clients_num;
        printf("저런! 용사 1명이 낙오됐어요! 현재 %d명!\n", clients_num);
    pthread_mutex_unlock(&mutex);

    close(*(int*)client_socket);

    if(!clients_num) fclose(chat_log);
}
```

- **사용자 정의 함수 목록(계속)**
  - manage_client(): 길이 0의 Message를 송신한 Client의 연결 해제, Client들의 총원 감소, SD 자리 비우기

5. Chatting

채팅 기능

```
int main(int argc, char* argv[]) {
    int server_socket, client_socket;
    struct sockaddr_in server_address, client_address;
    int client_address_size= sizeof(client_address);
    pthread_t clients_t[100];

    pthread_mutex_init(&mutex, NULL);
    server_socket= socket(PF_INET, SOCK_STREAM, 0);
    server_address.sin_family= AF_INET;
    server_address.sin_addr.s_addr= htonl(INADDR_ANY);
    server_address.sin_port= htons(atoi(argv[1]));

    if(bind(server_socket, (struct sockaddr*)&server_address, sizeof(server_address)) == -1) {
        printf("Failed to bind address to socket.\n");
        exit(1);
    }

    if(listen(server_socket, 10) == -1) {
        printf("앗! 우당탕 사고 발생!\n");
        exit(1);
    }

    chat_log= fopen("chat_log.txt", "a");
    printf("문을 열었어요! 오늘은 어떤 모험가가 사고를 칠까요? ></\n");
```

- **진입점 (main())**
  - 각종 변수를 초기화하고 주소의 Binding 작업 후 대기 상태에 돌입

5. Chatting

채팅 기능

```
    chat_log= fopen("chat_log.txt", "a");
    printf("문을 열었어요! 오늘은 어떤 모험가가 사고를 칠까요? ></\n");

    while(1) {
        client_socket= accept(server_socket, (struct sockaddr*)&client_address, &client_address_size);

        pthread_mutex_lock(&mutex);
            int available= space_available();
            clients[available]= client_socket;

            printf("%d번 socket은 %d번에 할당했습니다.\n", client_socket, available);
            ++clients_num;
        pthread_mutex_unlock(&mutex);

        pthread_create(&clients_t[available], NULL, manage_client, (void*)&clients[available]);
        pthread_detach(clients_t[available]);

        printf("모험가 등장! 파티원은 %d명이랍니다. IP: %s\n", clients_num, inet_ntoa(client_address.sin_addr));
    }

    close(server_socket);
    return 0;
}
```

- **진입점 (계속)**
  - 내역 작성을 위한 File 개방
  - 연결 수락된 Client의 Socket을 빈 곳에 배정하고 각자를 Thread로 분리

5. Chatting

채팅 기능

```
char my_name[16];
char message[1008];
char server_port[8];
char server_IP[16];

void* send_message(void* server_socket) {
    char my_information[64];
    char name_message[1024];

    sprintf(my_information, "앗! 야생의 [%s]가 나타났다!\n", my_name);
    fflush(stdin);
    write(*(int*)server_socket, my_information, strlen(my_information));

    while(1) {
        fgets(message, 1008, stdin);
        fflush(stdin);

        if(!strncmp(message, "::quit", 6)) {
            write(*(int*)server_socket, "\0", 1);
            close(*(int*)server_socket);
            exit(0);
        }

        sprintf(name_message, "[%s]: %s", my_name, message);
        write(*(int*)server_socket, name_message, strlen(name_message)+1);
    }
}
```

Client

- **전역 변수 목록**
  - my_name: 사용할 이름
  - message: 작성한 Message
  - server_port: Server가 개방한 Port
  - server_IP: Server의 논리 주소(IPv4)
- **사용자 정의 함수 목록**
  - send_message(): 작성한 Message를 이름 정보와 결합해 Server로 전송(“::quit”를 입력하면 길이 0의 Message로 가공되어 Server로 전송됨)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <pthread.h>
```

5. Chatting

채팅 기능

```c
void* receive_message(void* server_socket) {
    char name_message[1024];
    int message_length;

    while(1) {
        message_length= read(*(int*)server_socket, name_message, 1023);
        if(message_length == -1)
            return (void*)-1;

        name_message[message_length]= '\0';
        fputs(name_message, stdout);
    }
}


int main(int argc, char *argv[]) {
    int server_socket;
    struct sockaddr_in server_address;
    pthread_t send_t, receive_t;
    void* t_terminate;

    sprintf(server_IP, "%s", argv[1]);
    sprintf(server_port, "%s", argv[2]);
    sprintf(my_name, "%s", argv[3]);
    if((server_socket= socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        printf("Failed to create socket.\n");
        exit(1);
    }

    printf("어디로든 문!\n");

    server_address.sin_family= AF_INET;
    server_address.sin_addr.s_addr= inet_addr(server_IP);
    server_address.sin_port= htons(atoi(server_port));
    if(connect(server_socket, (struct sockaddr*)&server_address, sizeof(server_address)) == -1) {
        printf("앗! 너무 늦었네요! 파티원들은 이미 모험을 떠났답니다! :/\n");
        exit(1);
    }

    pthread_create(&send_t, NULL, send_message, (void*)&server_socket);
    pthread_create(&receive_t, NULL, receive_message, (void*)&server_socket);
    pthread_join(send_t, &t_terminate);
    pthread_join(receive_t, &t_terminate);

    close(server_socket);
    return 0;
}
```

- **사용자 정의 함수 목록(계속)**
  - receive_message(): Server에서 자신의 Socket Descriptor를 참조해 기록한 Message 가져오기
- **진입점**
  - 각종 변수 초기화
  - Server에 연결 요청 후 수락 시 송수신 함수 각각을 Thread로 분할하여 수행

6. File Transmitting

파일 전송 기능

```c
int main(int argc, char* argv[]) {

    int my_socket, client_socket;
    struct sockaddr_in my_address, client_address;

    char ibuf[1024]= {0};
    FILE* file;

    my_address.sin_family= AF_INET;
    my_address.sin_addr.s_addr= INADDR_ANY;
    my_address.sin_port= htons(atoi(argv[2]));

    if((my_socket= socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket 생성 실패\n");
        exit(EXIT_FAILURE);
    }

    printf("Socket 생성 성공\n");

    if(bind(my_socket, (struct sockaddr*)&my_address, sizeof(my_address)) < 0) {
        perror("Network 주소 할당 실패\n");
        close(my_socket);
        exit(EXIT_FAILURE);
    }

    printf("Server 주소 할당 성공\n");
    char tmp[32];
    inet_ntop(AF_INET, &my_address.sin_addr.s_addr, tmp, sizeof(tmp));
    printf("IP: %s\n", tmp);
    printf("Port: %hu\n", ntohs(my_address.sin_port));

    if(listen(my_socket, 3) < 0) {
        perror("Client의 요청 수행 불가\n");
        close(my_socket);
        exit(EXIT_FAILURE);
    }
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```c
    printf("Client의 요청 대기\n");
    int size_client_address= sizeof(client_address);
    client_socket= accept(my_socket, (struct sockaddr*)&client_address, (socklen_t*)&client_address);
    if(client_socket < 0) {
        perror("Client의 무효한 요청\n");
        close(my_socket);
        exit(EXIT_FAILURE);
    }

    printf("Client와 연결 성공\n");

    file= fopen(argv[1], "rb");
    if(file == NULL) {
        perror("File 열람 불가\n");
        close(client_socket);
        close(my_socket);
        exit(EXIT_FAILURE);
    }

    int B_read;
    while((B_read= fread(ibuf, 1, 1024, file)) > 0) {
        send(client_socket, ibuf, B_read, 0);
        memset(ibuf, 0, 1024);
    }

    printf("%s 전송 완료\n", argv[1]);

    fclose(file);
    close(client_socket);
    close(my_socket);

    return 0;
}
```

6. File Transmitting

파일 전송 기능

```c
int main(int argc, char* argv[]) {
    int my_socket;
    struct sockaddr_in server_address;

    char ibuf[1024]= {0};
    FILE* file;

    if((my_socket= socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket 생성 실패\n");
        return -1;
    }

    server_address.sin_family= AF_INET;
    server_address.sin_port= htons(atoi(argv[2]));
    server_address.sin_addr.s_addr= inet_addr(argv[1]);
    if(inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {
        perror("무효한 IP 주소\n");
        return -1;
    }

    if(connect(my_socket, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {
        perror("Server 연결 실패\n");
        return -1;
    }

    printf("Server 연결 성공\n");
    printf("Port: %hu\n", ntohs(server_address.sin_port));
```

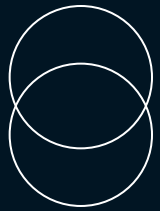
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
```

```c
    file= fopen("received", "wb");
    if(file == NULL) {
        perror("File 열람 실패\n");
        close(my_socket);
        return -1;
    }

    int B_receive;
    while((B_receive= recv(my_socket, ibuf, 1024, 0)) > 0) {
        fwrite(ibuf, 1, B_receive, file);
        memset(ibuf, 0, 1024);
    }

    printf("File 수신 완료\n", ibuf);

    fclose(file);
    close(my_socket);

    return 0;
}
```

7. Role

역할분담

이름	역할	비고
송민우	전체 파일 통합 및 각 파일 버그 수정, 뽀모도로 타이머 구현, git파일 업로드	
김건희	채팅 및 파일 전송 서비스 내부 구현	
최원서	채팅, 파일 업로드,검색 기능 구현	
김재혁	캘린더 인터페이스 제작, 캘린더 시스템 구현	
이채민	뽀모도로 타이머 세부 함수 제작	
손민승	캘린더 수정, 메인인터페이스 수정 및 구현	



Thank You

5조 송민우, 김건희, 최원서, 김재혁, 이채민, 손민승