**Priority 및 RR 스케줄링 알고리즘 구현**

컴퓨터공학과

2142851 김형준

운영체제 02분반

**목차**

1. **Nonpreemptive\_Priority**
   1. 소스 코드
   2. 결과 화면
2. **Preemptive\_Priority**
   1. 소스 코드
   2. 결과 화면
3. **Round\_Robin**
   1. 소스 코드
   2. 결과 화면
4. **참고자료**
5. **Nonpreemptive\_Priority**
   1. **소스 코드**

*// 2142851 컴퓨터공학과 김형준*

*// Nonpreemptive\_Priority*

*#include* <stdio.h>

*#include* <pthread.h>

*#include* <stdlib.h>

*#include* <unistd.h>

*#include* <cstring>

*#define* THREAD\_COUNT 5

char gantt\_chart[300];

static int fixed\_running\_time[THREAD\_COUNT] = { 10, 28, 6, 4, 14 };

static int fixed\_starting\_time[THREAD\_COUNT] = { 0, 1, 2, 3, 4 };

int priority[THREAD\_COUNT] = { 3, 2, 4, 1, 2 };

int return\_time[THREAD\_COUNT] = { 0, }; *// 반환 시간*

int waiting\_time[THREAD\_COUNT] = { 0, }; *// 대기 시간*

int total\_time = 0; *// 전체 시간*

int next\_processing\_number[THREAD\_COUNT] = { 1, 1, 1, 1, 1 }; *// 다음으로 실행할 시간*

int previous\_time[THREAD\_COUNT]; *// 이전까지 실행한 시간*

typedef struct Process {

    int id; *// Process ID*

    int multiplier; *// n X multiplier*

    int running\_time; *// Process Run Time*

    int priority;

    struct Process\* next;

} Process;

typedef struct Queue {

    Process\* front;

    Process\* rear;

    pthread\_mutex\_t lock;

    pthread\_cond\_t not\_empty;

} Queue;

*// print*

void printQueue(Queue\* *q*) {

    pthread\_mutex\_lock(&q->lock);

    Process\* current = q->front;

    printf("============================ <Queue> ============================\n");

*if* (current == NULL) printf("|\t\t\tCLEAR\t\t\t\t\t|\n");

*while* (current != NULL) {

        printf("|\tProcess ID: %d, Multiplier: %d, Running Time: %d\t\t|\n", current->id, current->multiplier, current->running\_time);

        current = current->next;

    }

    printf("=================================================================\n");

    pthread\_mutex\_unlock(&q->lock);

}

*// init*

void initQueue(Queue\* *q*) {

    q->front = q->rear = NULL;

    pthread\_mutex\_init(&q->lock, NULL);

    pthread\_cond\_init(&q->not\_empty, NULL);

}

void enqueue(Queue\* *q*, Process\* *process*) {

    pthread\_mutex\_lock(&q->lock);

*if* (q->rear == NULL) {

        q->front = q->rear = process;

    }

*else* {

        Process\* current = q->front;

*// sort loop*

*while* (current->next != NULL && current->next->priority >= process->priority) {

*// current를 현재의 다음으로 설정*

            current = current->next;

        }

        process->next = current->next;

        current->next = process;

*// 삽입된 process의 다음이 NULL일 경우*

*// (마지막 순번이라면)*

*if* (process->next == NULL) {

            q->rear = process;

        }

    }

    pthread\_cond\_signal(&q->not\_empty);

    pthread\_mutex\_unlock(&q->lock);

}

Process\* dequeue(Queue\* *q*) {

    pthread\_mutex\_lock(&q->lock);

*while* (q->front == NULL) {

        pthread\_cond\_wait(&q->not\_empty, &q->lock);

    }

    Process\* process = q->front;

    q->front = q->front->next;

*if* (q->front == NULL) {

        q->rear = NULL;

    }

    pthread\_mutex\_unlock(&q->lock);

*return* process;

}

void\* processThread(void\* *arg*) {

    Queue\* q = (Queue\*)arg;

    Process\* process = dequeue(q);

    pthread\_mutex\_lock(&q->lock); *// lock*

    sprintf(gantt\_chart + strlen(gantt\_chart), "P%d (%d-", process->id, total\_time);

    int start\_time = total\_time;

    previous\_time[process->id - 1] = next\_processing\_number[process->id - 1] - 1;

*for* (int i = next\_processing\_number[process->id - 1]; i <= process->running\_time; i++) {

        usleep(10000); *// 0.01 second delay*

        printf("[TIME: %2d] P%d: %2d X %2d = %2d\n", total\_time, process->id, i, process->multiplier, i \* process->multiplier);

        total\_time++;

*// printf("total\_time : %d\n", total\_time);*

        next\_processing\_number[process->id - 1] = i + 1;

    }

    return\_time[process->id - 1] = total\_time - fixed\_starting\_time[process->id - 1];

*// 대기시간 = 마지막 작업 시작 시간(start\_time) - 도착 시간(fixed\_starting\_time) - 이전 실행 시간의 합(previous\_time)*

    waiting\_time[process->id - 1] = start\_time - fixed\_starting\_time[process->id - 1] - previous\_time[process->id - 1];

    sprintf(gantt\_chart + strlen(gantt\_chart), "%d)\n", total\_time);

    pthread\_mutex\_unlock(&q->lock); *// unlock*

    free(process);

*return* NULL;

}

int main() {

    Queue q;

    initQueue(&q);

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        Process\* process = (Process\*)malloc(sizeof(Process));

        process->id = i + 1;

        process->multiplier = i + 1;

        process->next = NULL;

        process->running\_time = fixed\_running\_time[i];

        process->priority = priority[i];

        enqueue(&q, process);

        printQueue(&q);

    }

    pthread\_t threads[THREAD\_COUNT];

*for* (int i = 0; i < THREAD\_COUNT; i++) {

*while* (total\_time < fixed\_starting\_time[i]);

        printf("[TIME: %2d] P%d is arrived\n", total\_time, i + 1);

        pthread\_create(&threads[i], NULL, processThread, &q);

    }

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        pthread\_join(threads[i], NULL);

    }

    printf("END\n");

    printQueue(&q);

    printf("===========================================================\n");

    printf("Process\t|\tReturn Time\t|\tWaiting Time\n");

    printf("-----------------------------------------------------------\n");

    double sum\_return\_time = 0, sum\_waiting\_time = 0;

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        printf("P%d\t|\t%d\t\t|\t%d\n", i + 1, return\_time[i], waiting\_time[i]);

        sum\_return\_time += return\_time[i];

        sum\_waiting\_time += waiting\_time[i];

    }

    printf("===========================================================\n");

    printf("Result\t|\t평균 반환시간\t|\t평균 대기시간\n");

    printf("-----------------------------------------------------------\n");

    printf("-\t|\t%.1lf\t\t|\t%.1lf\n", sum\_return\_time / THREAD\_COUNT, sum\_waiting\_time / THREAD\_COUNT);

    printf("===========================================================\n");

    printf("Gantt Chart\n");

    printf("-----------------------------------------------------------\n");

    printf("%s", gantt\_chart);

    printf("===========================================================\n");

*return* 0;

}

* 1. **결과 화면**

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. **비선점(Nonpreemptive) SJF 구현**
   1. **소스 코드**

// 2142851 컴퓨터공학과 김형준

// Round-Robin

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

#include <cstring>

#define THREAD\_COUNT 5

char gantt\_chart[300];

static int fixed\_running\_time[THREAD\_COUNT] = { 10, 28, 6, 4, 14 };

static int fixed\_starting\_time[THREAD\_COUNT] = { 0, 1, 2, 3, 4 };

int priority[THREAD\_COUNT] = { 3, 2, 4, 1, 2 };

int return\_time[THREAD\_COUNT] = { 0, };     // 반환 시간

int waiting\_time[THREAD\_COUNT] = { 0, };    // 대기 시간

int total\_time = 0;                         // 전체 시간

int next\_processing\_number[THREAD\_COUNT] = { 1, 1, 1, 1, 1 };   // 다음으로 실행할 시간

int previous\_time[THREAD\_COUNT];                                // 이전까지 실행한 시간

int now = 0;

int previous = 0;

typedef struct Process {

    int id;                 // Process ID

    int multiplier;         // n X multiplier

    int running\_time;       // Process Run Time

    struct Process\* next;

} Process;

typedef struct Queue {

    Process\* front;

    Process\* rear;

    pthread\_mutex\_t lock;

    pthread\_cond\_t not\_empty;

} Queue;

// print

void printQueue(Queue\* q) {

    pthread\_mutex\_lock(&q->lock);

    Process\* current = q->front;

    printf("============================ <Queue> ============================\n");

    if (current == NULL) printf("|\t\t\tCLEAR\t\t\t\t\t|\n");

    while (current != NULL) {

        printf("|\tProcess ID: %d, Multiplier: %d, Running Time: %d\t\t|\n", current->id, current->multiplier, current->running\_time);

        current = current->next;

    }

    printf("=================================================================\n");

    pthread\_mutex\_unlock(&q->lock);

}

// init

void initQueue(Queue\* q) {

    q->front = q->rear = NULL;

    pthread\_mutex\_init(&q->lock, NULL);

    pthread\_cond\_init(&q->not\_empty, NULL);

}

void enqueue(Queue\* q, Process\* process) {

    pthread\_mutex\_lock(&q->lock);

    if (q->rear == NULL) {

        q->front = q->rear = process;

    } else {

        q->rear->next = process;

        q->rear = process;

    }

    pthread\_cond\_signal(&q->not\_empty);

    pthread\_mutex\_unlock(&q->lock);

}

Process\* dequeue(Queue\* q) {

    pthread\_mutex\_lock(&q->lock);

    while (q->front == NULL) {

        pthread\_cond\_wait(&q->not\_empty, &q->lock);

    }

    Process\* process = q->front;

    q->front = q->front->next;

    if (q->front == NULL) {

        q->rear = NULL;

    }

    pthread\_mutex\_unlock(&q->lock);

    return process;

}

int run\_arrow = 0;

bool flag[THREAD\_COUNT] = { true, false, false, false, false };

bool is\_end[THREAD\_COUNT] = { false, false, false, false, false };

int select\_arrow() {

    int max = 1;

    int id;

    int cnt = 0;

    for (int i = 0; i < THREAD\_COUNT; i++) {

        if (is\_end[i] == true) {

            cnt++;

            continue;

        }

        id = (max < priority[i]) ? i : id;

        max = (max < priority[i]) ? priority[i] : max;

        printf("max: %d | priority[%d]: %d\n", max, i, priority[i]);

        printf("id: %d\n", id);

    }

    if (cnt == 4) {

        for (int i = 0; i < THREAD\_COUNT; i++) {

            if (is\_end[i] == false) {

                id = i;

                break;

            }

        }

    }

    return id;

}

bool stop\_flag[THREAD\_COUNT] = { false, false, false, false, false };

void\* processThread(void\* arg) {

    Queue\* q = (Queue\*)arg;

    Process\* process = dequeue(q);

    while (!flag[process->id - 1]);

    pthread\_mutex\_lock(&q->lock); // lock

    now = process->id - 1;

    sprintf(gantt\_chart + strlen(gantt\_chart), "P%d (%d-", process->id, total\_time);

    int start\_time = total\_time;

    previous\_time[process->id - 1] = 0;

    int running = 0;

    for (int i = next\_processing\_number[process->id - 1]; i <= process->running\_time; i++) {

        // printf("[TIME: %2d] P%d: STOPFLAG: %d\n", total\_time, process->id, stop\_flag[process->id - 1]);

        usleep(10000); // 0.01 second delay

        printf("[TIME: %2d] P%d: %2d X %2d = %2d\n", total\_time, process->id, i, process->multiplier, i \* process->multiplier);

        // printf("[TIME: %2d] P%d: STOPFLAG: %d\n", total\_time, process->id, stop\_flag[process->id - 1]);

        total\_time++;

        // printf("total\_time : %d\n", total\_time);

        next\_processing\_number[process->id - 1] = i + 1;

        running++;

        // getchar();

        usleep(10000); // 0.01 second delay

        if (stop\_flag[process->id - 1] || i == process->running\_time) {

            previous = process->id - 1;

            flag[process->id - 1] = false;

            if (i == process->running\_time) {

                // printf("[TIME: %2d] P%d END\n", total\_time, process->id);

                // printf("[TIME: %2d] select\_arrow: %d (P%d)\n", total\_time, run\_arrow, run\_arrow + 1);

                is\_end[process->id - 1] = true;

                int n = select\_arrow();

                printf("NEXT: P%d\n", n + 1);

                flag[n] = true;

                // flag[previous] = true;

                break;

            }

            previous\_time[process->id - 1] += running;

            sprintf(gantt\_chart + strlen(gantt\_chart), "%d)\n", total\_time);

            pthread\_mutex\_unlock(&q->lock); // unlock

            while (!flag[process->id - 1]);

            pthread\_mutex\_lock(&q->lock); // lock

            stop\_flag[process->id - 1] = false;

            now = process->id - 1;

            running = 0;

            start\_time = total\_time;

            sprintf(gantt\_chart + strlen(gantt\_chart), "P%d (%d-", process->id, total\_time);

        }

        // printf("[TIME: %2d] P%d: STOPFLAG: %d\n", total\_time, process->id, stop\_flag[process->id - 1]);

    }

    return\_time[process->id - 1] = total\_time - fixed\_starting\_time[process->id - 1];

    // 대기시간 = 마지막 작업 시작 시간(start\_time) - 도착 시간(fixed\_starting\_time) - 이전 실행 시간의 합(previous\_time)

    // printf("=========================================================\n");

    // printf("start\_time: %d\n", start\_time);

    // printf("fixed\_starting\_time[%d]: %d\n", process->id - 1, fixed\_starting\_time[process->id - 1]);

    // printf("previous\_time[%d]: %d\n", process->id - 1, previous\_time[process->id - 1]);

    // printf("=========================================================\n");

    waiting\_time[process->id - 1] = start\_time - fixed\_starting\_time[process->id - 1] - previous\_time[process->id - 1];

    sprintf(gantt\_chart + strlen(gantt\_chart), "%d)\n", total\_time);

    pthread\_mutex\_unlock(&q->lock); // unlock

    free(process);

    return NULL;

}

int main() {

    Queue q;

    initQueue(&q);

    for (int i = 0; i < THREAD\_COUNT; i++) {

        Process\* process = (Process\*)malloc(sizeof(Process));

        process->id = i + 1;

        process->multiplier = i + 1;

        process->next = NULL;

        process->running\_time = fixed\_running\_time[i];

        enqueue(&q, process);

        printQueue(&q);

    }

    pthread\_t threads[THREAD\_COUNT];

    for (int i = 0; i < THREAD\_COUNT; i++) {

        while (total\_time != fixed\_starting\_time[i]);

        printf("[TIME: %2d] P%d is arrived\n", total\_time, i + 1);

        if (priority[now] < priority[i]) {

            // getchar();

            stop\_flag[now] = true;

            printf("[TIME: %2d] P%d is in\n", total\_time, i + 1);

            // getchar();

            flag[i] = true;

        }

        pthread\_create(&threads[i], NULL, processThread, &q);

    }

    for (int i = 0; i < THREAD\_COUNT; i++) {

        pthread\_join(threads[i], NULL);

    }

    printf("END\n");

    printQueue(&q);

    printf("===========================================================\n");

    printf("Process\t|\tReturn Time\t|\tWaiting Time\n");

    printf("-----------------------------------------------------------\n");

    double sum\_return\_time = 0, sum\_waiting\_time = 0;

    for (int i = 0; i < THREAD\_COUNT; i++) {

        printf("P%d\t|\t%d\t\t|\t%d\n", i + 1, return\_time[i], waiting\_time[i]);

        sum\_return\_time += return\_time[i];

        sum\_waiting\_time += waiting\_time[i];

    }

    printf("===========================================================\n");

    printf("Result\t|\t평균 반환시간\t|\t평균 대기시간\n");

    printf("-----------------------------------------------------------\n");

    printf("-\t|\t%.1lf\t\t|\t%.1lf\n", sum\_return\_time / THREAD\_COUNT, sum\_waiting\_time / THREAD\_COUNT);

    printf("===========================================================\n");

    printf("Gantt Chart\n");

    printf("-----------------------------------------------------------\n");

    printf("%s", gantt\_chart);

    printf("===========================================================\n");

    return 0;

}

* 1. **결과 화면**

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

1. **Round\_Robin**
   1. **소스 코드**

*// 2142851 컴퓨터공학과 김형준*

*// Round-Robin*

*#include* <stdio.h>

*#include* <pthread.h>

*#include* <stdlib.h>

*#include* <unistd.h>

*#include* <cstring>

*#define* THREAD\_COUNT 5

char gantt\_chart[300];

static int fixed\_running\_time[THREAD\_COUNT] = { 10, 28, 6, 4, 14 };

static int fixed\_starting\_time[THREAD\_COUNT] = { 0, 1, 2, 3, 4 };

int return\_time[THREAD\_COUNT] = { 0, }; *// 반환 시간*

int waiting\_time[THREAD\_COUNT] = { 0, }; *// 대기 시간*

int total\_time = 0; *// 전체 시간*

int next\_processing\_number[THREAD\_COUNT] = { 1, 1, 1, 1, 1 }; *// 다음으로 실행할 시간*

int previous\_time[THREAD\_COUNT]; *// 이전까지 실행한 시간*

typedef struct Process {

    int id; *// Process ID*

    int multiplier; *// n X multiplier*

    int running\_time; *// Process Run Time*

    struct Process\* next;

} Process;

typedef struct Queue {

    Process\* front;

    Process\* rear;

    pthread\_mutex\_t lock;

    pthread\_cond\_t not\_empty;

} Queue;

*// print*

void printQueue(Queue\* *q*) {

    pthread\_mutex\_lock(&q->lock);

    Process\* current = q->front;

    printf("============================ <Queue> ============================\n");

*if* (current == NULL) printf("|\t\t\tCLEAR\t\t\t\t\t|\n");

*while* (current != NULL) {

        printf("|\tProcess ID: %d, Multiplier: %d, Running Time: %d\t\t|\n", current->id, current->multiplier, current->running\_time);

        current = current->next;

    }

    printf("=================================================================\n");

    pthread\_mutex\_unlock(&q->lock);

}

*// init*

void initQueue(Queue\* *q*) {

    q->front = q->rear = NULL;

    pthread\_mutex\_init(&q->lock, NULL);

    pthread\_cond\_init(&q->not\_empty, NULL);

}

void enqueue(Queue\* *q*, Process\* *process*) {

    pthread\_mutex\_lock(&q->lock);

*if* (q->rear == NULL) {

        q->front = q->rear = process;

    } *else* {

        q->rear->next = process;

        q->rear = process;

    }

    pthread\_cond\_signal(&q->not\_empty);

    pthread\_mutex\_unlock(&q->lock);

}

Process\* dequeue(Queue\* *q*) {

    pthread\_mutex\_lock(&q->lock);

*while* (q->front == NULL) {

        pthread\_cond\_wait(&q->not\_empty, &q->lock);

    }

    Process\* process = q->front;

    q->front = q->front->next;

*if* (q->front == NULL) {

        q->rear = NULL;

    }

    pthread\_mutex\_unlock(&q->lock);

*return* process;

}

int run\_arrow = 0;

bool flag[THREAD\_COUNT] = { true, false, false, false, false };

bool is\_end[THREAD\_COUNT] = { false, false, false, false, false };

void select\_arrow(int *id*) {

*// printf("[TIME: %2d] SELECT ARROW FUNCTION START\n", total\_time);*

    int num;

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        num = (id + i) % THREAD\_COUNT;

*// printf("[TIME: %2d] i: %d\n", total\_time, i);*

*// printf("[TIME: %2d] id: %d\n", total\_time, id);*

*// printf("[TIME: %2d] num: %d\n", total\_time, num);*

*if* (is\_end[num]) {

*continue*;

        }

*else* {

*break*;

        }

    }

    run\_arrow = num;

*// printf("[TIME: %2d] run\_arrow: %d\n", total\_time, run\_arrow);*

    flag[run\_arrow] = true;

}

void\* processThread(void\* *arg*) {

    Queue\* q = (Queue\*)arg;

    Process\* process = dequeue(q);

    pthread\_mutex\_lock(&q->lock); *// lock*

    sprintf(gantt\_chart + strlen(gantt\_chart), "P%d (%d-", process->id, total\_time);

    int start\_time = total\_time;

    previous\_time[process->id - 1] = 0;

    int running = 0;

*for* (int i = next\_processing\_number[process->id - 1]; i <= process->running\_time; i++) {

        usleep(10000); *// 0.01 second delay*

        printf("[TIME: %2d] P%d: %2d X %2d = %2d\n", total\_time, process->id, i, process->multiplier, i \* process->multiplier);

        total\_time++;

*// printf("total\_time : %d\n", total\_time);*

        next\_processing\_number[process->id - 1] = i + 1;

        running++;

*if* (running == 5 || i == process->running\_time) {

            flag[process->id - 1] = false;

            select\_arrow(process->id);

*if* (i == process->running\_time) {

*// printf("[TIME: %2d] P%d END\n", total\_time, process->id);*

*// printf("[TIME: %2d] select\_arrow: %d (P%d)\n", total\_time, run\_arrow, run\_arrow + 1);*

                is\_end[process->id - 1] = true;

*break*;

            }

            previous\_time[process->id - 1] += running;

            sprintf(gantt\_chart + strlen(gantt\_chart), "%d)\n", total\_time);

            pthread\_mutex\_unlock(&q->lock); *// unlock*

*while* (!flag[process->id - 1]);

            pthread\_mutex\_lock(&q->lock); *// lock*

            running = 0;

            start\_time = total\_time;

            sprintf(gantt\_chart + strlen(gantt\_chart), "P%d (%d-", process->id, total\_time);

        }

    }

    return\_time[process->id - 1] = total\_time - fixed\_starting\_time[process->id - 1];

*// 대기시간 = 마지막 작업 시작 시간(start\_time) - 도착 시간(fixed\_starting\_time) - 이전 실행 시간의 합(previous\_time)*

*// printf("=========================================================\n");*

*// printf("start\_time: %d\n", start\_time);*

*// printf("fixed\_starting\_time[%d]: %d\n", process->id - 1, fixed\_starting\_time[process->id - 1]);*

*// printf("previous\_time[%d]: %d\n", process->id - 1, previous\_time[process->id - 1]);*

*// printf("=========================================================\n");*

    waiting\_time[process->id - 1] = start\_time - fixed\_starting\_time[process->id - 1] - previous\_time[process->id - 1];

    sprintf(gantt\_chart + strlen(gantt\_chart), "%d)\n", total\_time);

    pthread\_mutex\_unlock(&q->lock); *// unlock*

    free(process);

*return* NULL;

}

int main() {

    Queue q;

    initQueue(&q);

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        Process\* process = (Process\*)malloc(sizeof(Process));

        process->id = i + 1;

        process->multiplier = i + 1;

        process->next = NULL;

        process->running\_time = fixed\_running\_time[i];

        enqueue(&q, process);

        printQueue(&q);

    }

    pthread\_t threads[THREAD\_COUNT];

*for* (int i = 0; i < THREAD\_COUNT; i++) {

*while* (total\_time != fixed\_starting\_time[i]);

        printf("[TIME: %2d] P%d is arrived\n", total\_time, i + 1);

        pthread\_create(&threads[i], NULL, processThread, &q);

    }

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        pthread\_join(threads[i], NULL);

    }

    printf("END\n");

    printQueue(&q);

    printf("===========================================================\n");

    printf("Process\t|\tReturn Time\t|\tWaiting Time\n");

    printf("-----------------------------------------------------------\n");

    double sum\_return\_time = 0, sum\_waiting\_time = 0;

*for* (int i = 0; i < THREAD\_COUNT; i++) {

        printf("P%d\t|\t%d\t\t|\t%d\n", i + 1, return\_time[i], waiting\_time[i]);

        sum\_return\_time += return\_time[i];

        sum\_waiting\_time += waiting\_time[i];

    }

    printf("===========================================================\n");

    printf("Result\t|\t평균 반환시간\t|\t평균 대기시간\n");

    printf("-----------------------------------------------------------\n");

    printf("-\t|\t%.1lf\t\t|\t%.1lf\n", sum\_return\_time / THREAD\_COUNT, sum\_waiting\_time / THREAD\_COUNT);

    printf("===========================================================\n");

    printf("Gantt Chart\n");

    printf("-----------------------------------------------------------\n");

    printf("%s", gantt\_chart);

    printf("===========================================================\n");

*return* 0;

}

* 1. **결과 화면**

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명