**FCFS 및 SJF 구현**

컴퓨터공학과

2142851 김형준

운영체제 02분반

**목차**

1. **FCFS 구현**
   1. 소스 코드
   2. 결과 화면
2. **비선점(Nonpreemptive) SJF 구현**
   1. 소스 코드
   2. 결과 화면
3. **선점(Preemptive) SJF 구현**
   1. 소스 코드
   2. 결과 화면
   3. 문제점
4. **참고자료**
5. **FCFS 구현**
   1. **소스 코드**

*// 2142851 컴퓨터공학과 김형준*

*// FCFS*

*#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;

}

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];

        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*

*// Nonpreemptive\_SJF*

*#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* {

        Process\* current = q->front;

*// current->next의 running\_time이 process의 running\_time보다 짧을 경우까지 반복*

*while* (current->next != NULL && current->next->running\_time < process->running\_time) {

*// 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];

        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. **선점(Preemptive) SJF 구현**
   1. **소스 코드**

// 2142851 컴퓨터공학과 김형준

// Preemptive\_SJF

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#include <unistd.h>

#include <cstring>

#define THREAD\_COUNT 5

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

char gantt\_chart[300];

int now = 0;

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;

Queue all\_q;

// 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;

        // current->next의 running\_time이 process의 running\_time보다 짧을 경우까지 반복

        while (current->next != NULL && current->next->running\_time < process->running\_time) {

            // 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;

}

// 큐에서 맨 앞에 있는 요소의 정보를 반환하는 함수

Process\* peek(Queue\* q) {

    pthread\_mutex\_lock(&q->lock);

    Process\* front = q->front;

    pthread\_mutex\_unlock(&q->lock);

    return front;

}

bool is\_arrived = false;

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

bool thread\_add\_flag = false;

bool end\_flag = false;

void\* processThread(void\* arg) {

    Queue\* q = (Queue\*)arg;

    Process\* process = dequeue(q);

    // while (flag[process->id - 1] == false) {

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

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

    //     if (flag[process->id - 1]) {

    //         break;

    //     }

    // }

    while (!flag[process->id - 1]);

    usleep(100000); // 0.1 second delay

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

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

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

    int start\_time = total\_time;

    bool is\_wait = false;

    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

        if (is\_arrived) {

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

            usleep(50000); // 0.05 second delay

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

        }

        usleep(150000); // 0.15 second delay

        printf("[TIME: %2d] [FLAG[%d]]: %d\n", total\_time, process->id - 1, flag[process->id - 1]);

        // getchar();

        if (!flag[process->id - 1]) {

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

            break;

        }

        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;

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

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

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

            thread\_add\_flag = true;

            if (q->front != NULL) {

                flag[q->front->id - 1] = true;

            }

        }

    }

    if (is\_done[process->id - 1] == false) {

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

        // printQueue(&all\_q);

        // enqueue(&all\_q, process);

        // printQueue(&all\_q);

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

    }

    end\_flag = true;

    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);

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

    printf("<gantt\_chart>\n%s\n", gantt\_chart);

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

    free(process);

    return NULL;

}

int main() {

    initQueue(&all\_q);

    pthread\_t threads[THREAD\_COUNT];

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

        printf("total\_time: %d\nfixed\_starting\_time[%d]: %d\n", total\_time, i, fixed\_starting\_time[i]);

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

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

        is\_arrived = true;

        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(&all\_q, process);

        // printf("1\n");

        printQueue(&all\_q);

        // printf("2\n");

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

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

        // Preemptive

        if (fixed\_running\_time[i] < fixed\_running\_time[now]) {

            printf("[TIME: %2d] [IF] fixed\_running\_time[P%d](%d) < fixed\_running\_time[P%d](%d)\n", total\_time, i + 1, fixed\_running\_time[i], now + 1, fixed\_running\_time[now]);

            flag[now] = false;

            flag[i] = true;

            printf("[TIME: %2d] [FLAG] now run: P%d, stop run: P%d\n", total\_time, i + 1, now + 1);

            // printf("<gantt\_chart>\n%s\n", gantt\_chart);

            now = i;

            // printQueue(&all\_q);

        }

    }

    while (1) {

        while(!end\_flag);

        printf("[TIME: %2d] [END\_FLAG is TRUE]\n", total\_time);

        int done\_cnt = 0;

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

            done\_cnt += is\_done[i];

        }

        if (done\_cnt == THREAD\_COUNT) {

            break;

        }

        usleep(10000); // 0.01 second delay

        if (thread\_add\_flag == true) {

            // printf("1\n");

            // 현재 queue에 내용이 없기 때문에, 아래 부분에서 segmentation 오류가 생김

            pthread\_create(&threads[all\_q.front->id - 1], NULL, processThread, &all\_q);

            // printf("1\n");

            thread\_add\_flag = false;

        }

        end\_flag = false;

    }

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

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

    }

    printf("END\n");

    printQueue(&all\_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. **문제점**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

먼저, Queue에 Running Time에 맞추어 Process가 잘 들어간 것을 확인하였고, P1 실행 중 P3가 새치기하는 상황까지 나온 것을 확인했습니다. 하지만, P3이 실행되며, 1-4-3-5-2 였던 queue에서 P3 이전의 P1, P4가 사라졌습니다.

문제 해결을 위해 저는 다음과 같은 문제를 해결하려고 노력했습니다.

“Process가 끝나지 않았는데 queue에서 제거되는 현상”

이를 위해 P3을 실행한다면, queue의 front를 P3으로 변경하려는 시도를 하였습니다.

하지만, 큐의 특성 상 순서를 바꾸는 것은 의미도 없을뿐더러 잘 해결되지 않았습니다. 두번째로 시도한 방법은 어차피 enqueue 함수가 running\_time의 순서대로 넣는 것이기 때문에, thread의 함수가 실행될 때, dequeue 후 enqueue를 다시 진행하는 방법을 선택하였습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

몇 번의 시도 후 아래와 같이 큐에 다시 들어가게 될 때, Process의 id와 정보가 제대로 들어가지 않는 문제가 발생하였습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

enqueue 부분을 수정 후 이상한 값이 나오는 문제는 해결되었으나, P4 이후 P3이 다시 실행되는 부분에서 막혔습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

간트차트가 어느정도 잘 출력되는 것을 확인하였습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

하지만, Segmentation fault (core dumped) 에러가 발생하였습니다.

코드를 처음부터 총 4번 작성하여보았고, 결국 해결해내지 못했습니다.

마지막 시도는 queue에 넣음과 동시에 thread를 생성하고, 중단 시 thread 함수 내에서 멈추었던 이전 코드와 다르게, thread를 그냥 종료한 후 새 thread를 생성하는 시도였습니다. 마지막 오류는 결과화면 캡쳐와 같습니다.

문제점은 queue에 내용이 없는데, thread를 생성하려고 했기 때문입니다. 이를 해결하기 위해, Process의 작업 완료 여부를 체크하고, 작업이 완료되지 않았는데 thread를 종료할 경우, queue에 thread 함수 내에 있는 process 구조체 값을 넣으려 하였습니다. 하지만, total\_time이 3일 때, Process 4가 들어오게 되고, 이 때, queue에는 빠져나간 Process 3의 정보가 아닌, Process 4의 정보가 무한대로 들어가있었습니다. 이 부분을 해결하지 못하였습니다.

패턴, 의류, 패브릭, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. **참고자료**

**Queue 구현**

1. <https://devpluto.tistory.com/entry/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0-c%EC%96%B8%EC%96%B4%EB%A1%9C-%ED%81%90-%EC%9B%90%ED%98%95-%ED%81%90-%EA%B5%AC%ED%98%84%ED%95%98%EA%B8%B0>
2. <https://ksk9820.tistory.com/187>