**HW 6. MLFQ 및 HRN 스케줄링 알고리즘 구현**

컴퓨터공학과

2142851 김형준

운영체제 02분반

**목차**

1. **MLFQ**
   1. 소스 코드
   2. 결과 화면
2. **HRN**
   1. 소스 코드
   2. 결과 화면
3. **MLFQ**
   1. **소스 코드**

*#include* <stdio.h>

*#include* <stdlib.h>

*#include* <string.h>

*#include* <stdbool.h>

*#include* <unistd.h>

*#include* <sys/types.h>

*#include* <sys/wait.h>

*#define* PROCESS\_COUNT 3

*#define* Q1\_TIME 1

*#define* Q2\_TIME 2

*#define* Q3\_TIME 4

typedef struct {

    int id; *// ID*

    int burst\_time; *// 실행 시간*

    int remaining\_time; *// 남은 실행 시간*

    int arrival\_time; *// 도착 시간*

    int return\_time; *// 반환 시간 (종료 - 도착)*

    int waiting\_time; *// 대기 시간 (반환 - 실행)*

    int tmp;

} Process;

*/\* ========================================== Queue Func Start ========================================== \*/*

typedef struct {

    Process\* queue[PROCESS\_COUNT];

    int front;

    int rear;

} Queue;

void initQueue(Queue\* *q*) {

    q->front = q->rear = -1;

}

bool isEmpty(Queue\* *q*) {

*return* q->front == -1;

}

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

*if* (q->rear == PROCESS\_COUNT - 1) *return*;

*if* (q->front == -1) q->front = 0;

    q->queue[++q->rear] = p;

}

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

*if* (isEmpty(q)) *return* NULL;

    Process\* p = q->queue[q->front];

*if* (q->front >= q->rear) q->front = q->rear = -1;

*else* q->front++;

*return* p;

}

*/\* ========================================== Queue Func End ========================================== \*/*

int tmp1 = 1;

int tmp2 = 1;

int tmp3 = 1;

*/\* ========================================== <MLFQ> Start ========================================== \*/*

void MLFQ(Process *processes*[], int *n*) {

    Queue q1, q2, q3;

    initQueue(&q1);

    initQueue(&q2);

    initQueue(&q3);

*for* (int i = 0; i < n; i++) {

        enqueue(&q1, &processes[i]);

    }

    int time = 0;

    char gantt\_chart[1000] = "";

    char tmp\_gantt\_chart[500] = "";

*while* (!isEmpty(&q1) || !isEmpty(&q2) || !isEmpty(&q3)) {

        Process\* current = NULL;

        int time\_quantum = 0;

*if* (!isEmpty(&q1)) {

            current = dequeue(&q1);

            time\_quantum = Q1\_TIME;

        } *else* *if* (!isEmpty(&q2)) {

            current = dequeue(&q2);

            time\_quantum = Q2\_TIME;

        } *else* *if* (!isEmpty(&q3)) {

            current = dequeue(&q3);

            time\_quantum = Q3\_TIME;

        }

*if* (current != NULL) {

*// if (current->remaining\_time > time\_quantum) {*

*//     exec\_time = time\_quantum;*

*// }*

*// else {*

*//     exec\_time = current->remaining\_time;*

*// }*

            int exec\_time = (current->remaining\_time > time\_quantum) ? time\_quantum : current->remaining\_time;

            sprintf(tmp\_gantt\_chart + strlen(tmp\_gantt\_chart), "P%d (%d-%d) ", current->id, time, time + exec\_time);

            time += exec\_time;

            current->remaining\_time -= exec\_time;

            printf("[TIME: %2d] [P%d] remaining\_time: %2d\n", time, current->id, current->remaining\_time);

*// fork!!*

            pid\_t pid = fork();

*if* (pid == 0) {

*switch* (current->id) {

*case* 1:

                        printf("[TIME: %2d] P%d: %d x %d = %d\n", time, current->*id*, current->*tmp*, current->*id*, current->tmp \* current->*id*);

                        current->tmp++;

*break*;

*case* 2:

                        printf("[TIME: %2d] P%d: %d x %d = %d\n", time, current->*id*, current->*tmp*, current->*id*, current->tmp \* current->*id*);

                        current->tmp++;

*break*;

*case* 3:

                        printf("[TIME: %2d] P%d: %d x %d = %d\n", time, current->*id*, current->*tmp*, current->*id*, current->tmp \* current->*id*);

                        current->tmp++;

*break*;

                }

                exit(0);

            } *else* *if* (pid > 0) {

                wait(NULL);

            }

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

*// printf("[TIME: %2d] Cycle clear\n", time);*

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

*if* (current->remaining\_time == 0) {

                current->return\_time = time - current->arrival\_time;

                current->waiting\_time = current->return\_time - current->burst\_time;

            } *else* {

*if* (current->remaining\_time > 1 && exec\_time == 1) {

                    sprintf(gantt\_chart + strlen(gantt\_chart), "Q1: %s\n", tmp\_gantt\_chart);

                    strcpy(tmp\_gantt\_chart, "");

                    enqueue(&q2, current);

                }

*else* *if* (current->remaining\_time > 2) {

                    sprintf(gantt\_chart + strlen(gantt\_chart), "Q2: %s\n", tmp\_gantt\_chart);

                    strcpy(tmp\_gantt\_chart, "");

                    enqueue(&q3, current);

                } *else* *if* (current->remaining\_time > 1) {

                    sprintf(gantt\_chart + strlen(gantt\_chart), "Q1: %s\n", tmp\_gantt\_chart);

                    strcpy(tmp\_gantt\_chart, "");

                    enqueue(&q2, current);

                } *else* {

                    sprintf(gantt\_chart + strlen(gantt\_chart), "Q3: %s\n", tmp\_gantt\_chart);

                    strcpy(tmp\_gantt\_chart, "");

                    enqueue(&q1, current);

                }

            }

        }

    }

    double total\_return\_time = 0;

    double total\_waiting\_time = 0;

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

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

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

*for* (int i = 0; i < n; i++) {

        printf("P%d\t|\t%d\t\t|\t%d\n", processes[i].id, processes[i].return\_time, processes[i].waiting\_time);

        total\_return\_time += processes[i].return\_time;

        total\_waiting\_time += processes[i].waiting\_time;

    }

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

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

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

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

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

    printf("Gantt Chart\n");

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

    printf("%s\n", gantt\_chart);

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

}

*/\* ========================================== <MLFQ> End ========================================== \*/*

int main() {

    Process processes[PROCESS\_COUNT] = {

        {1, 30, 30, 0, 0, 0, 1},

        {2, 20, 20, 0, 0, 0, 1},

        {3, 10, 10, 0, 0, 0, 1}

    };

    MLFQ(processes, PROCESS\_COUNT);

*return* 0;

}

* 1. **결과 화면**

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* 1. **Round-Robin과 성능 비교**

<RR의 결과 화면**>**

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

<반환시간 및 대기시간 비교 표>

|  |  |  |
| --- | --- | --- |
|  | **Round-Robin (RR)** | **MLFQ** |
| **반환시간** | 46.7 | 48.3 |
| **대기시간** | 26.7 | 28.3 |

<간트 차트>

**RR**

**스크린샷, 텍스트, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명**

**MLFQ**

텍스트, 스크린샷, 폰트, 대수학이(가) 표시된 사진

자동 생성된 설명

1. **HRN**
   1. **소스 코드**

*// 2142851 컴퓨터공학과 김형준*

*// HRN*

*#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;

}

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

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

void flag\_func() {

    int waited\_time[THREAD\_COUNT];

    int next\_id = 0;

    double value;

    int max = 0;

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

*if* (is\_end[i]) *continue*;

        waited\_time[i] = total\_time - fixed\_starting\_time[i];

        value = 1 + (1.0 \* waited\_time[i]/fixed\_running\_time[i]);

*if* (max < value) {

            max = value;

            next\_id = i;

        }

        printf("P%d: %lf\n", i + 1, value);

    }

    printf("next\_id: %d\n", next\_id);

    flag[next\_id] = true;

}

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

    Queue\* q = (Queue\*)arg;

    Process\* process = dequeue(q);

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

    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);

    is\_end[process->id - 1] = true;

    flag[process->id - 1] = false;

    flag\_func();

    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