
| 과목명 | 운영체제 |
|---|---|
| 담당교수 | 최종무 교수님 |
| 학과 | 소프트웨어학과 |
| 학번 | 32151671 / 32153180 |
| 이름 | 박민혁 / 이상민 |
| 제출일자 | 2019.04.16 |

# Ⅰ. 프로젝트 개요

## *1. 프로젝트 목표*

Scheduler algorithm인 FCFS, Round Robin, MLFQ를 구현할 수 있다.

## *2. 프로젝트 기본 개념*

FCFS(First Come First Serve)

: Schedule a process that arrives first

- advantages : clearly simple, easy to implement
- disadvantages : may cause a long waiting time (convoy effect)

RR(Round Robin)

: Instead of running a job to completion, it runs a job for a time slice and switch to the next job in the run queue
Repeatedly swtich jobs until jobs are finished

MLFQ(Multi-Level Feedback Queue)

: Consist of multiple queues and each queue is assigned a different priority level

Rule 1 : if Priority(A) > Priority(B), A runs (B doesn't)
Rule 2 : if Priority(A) = Priority(B), A & B run in RR
Rule 3 : when a job enters the system, it is placed at the highest priority (the topmost queue)
Rule 4a : if a job uses up an entire time slice while running, its priority is reduced (it moves down one queue)
Rule 4b : if a job gives up the CPU before the time slice is up, it stays at the same priority level
Rule 5 : after some time period S, move all the jobs in the system to the topmost queue (=Boost)

## Ⅱ. 프로젝트 시행

*1. 프로젝트 설계*

**lab1_sched.c**
**(schedular algorithm 관련 소스 파일)**

```
int zombie_q = 0;
                                // zombie_q 변수 초기화
int none = 0;
                                // none 변수 초기화

typedef struct process {
                // process 구조체 선언
        char process;
                                // process 이름
        int arrival_time;
                        // arrival time
        int service_time;
                                // service time
}process;

typedef struct ChainNode {
                        // ChainNode 구조체 선언
        char data;
                                        // data field
        struct ChainNode *link;
                        // link field
}ChainNode;

typedef struct Queue {
                        // Queue 구조체 선언
        char proc_name[6];
                                // 멤버변수 process name 배열 사용
}Queue;

typedef struct LinkedQueue {
                // LinkedQueue 구조체 선언
        ChainNode *front;
                                // Queue 제일 앞
        ChainNode *rear;
                                // Queue 제일 끝
        int count;
```

```
                                        // node 개수
}LinkedQueue;

process pr[5];
                                    // 구조체변수 pr 배열 선언
Queue q;
                                        // 구조체변수 q 선언
LinkedQueue *q1;
                                    // 구조체변수 포인터 선언
LinkedQueue *q2;
LinkedQueue *q3;
LinkedQueue *q4;

void initWorkload() {
                            // Workload 초기화
        pr[0].process = 'A';
        pr[0].arrival_time = 0;
        pr[0].service_time = 3;

        pr[1].process = 'B';
        pr[1].arrival_time = 2;
        pr[1].service_time = 6;

        pr[2].process = 'C';
        pr[2].arrival_time = 4;
        pr[2].service_time = 4;

        pr[3].process = 'D';
        pr[3].arrival_time = 6;
        pr[3].service_time = 5;

        pr[4].process = 'E';
        pr[4].arrival_time = 8;
        pr[4].service_time = 2;
}

void execute_RR();
                                    // execute_RR 함수 전방 선언
void queueShare(struct process p, int x) {
        // process 구조체 p와 x를 매개변수로 갖는 함수
```

```
        for (int i = 0; i < 5; i++) {
                // queueShare 함수는 FCFS와 RR에서 공동 사용
                if (q.proc_name[i] == '\0') {
                // process name 배열 비어있는 경우 추가
                        q.proc_name[i] = p.process;
                        if (x == 0)
                                        // FCFS인 경우 x = 1 / RR인 경우 x = 0
                                execute_RR();
                        break;
                }
        }
}

void execute_RR() {
        if (none != 1) {
                        // 큐가 죽지 않은 경우만 실행
                int count = 0;
                char tmp = q.proc_name[0];
                        // process name 첫번째 배열 tmp에 삽입

                for (int i = 0; i < 5; i++) {
                        if (q.proc_name[i] != '\0')
                        // process name 배열이 비어있지 않은 경우
                                count++;
                                        // 노드 개수 증가
                }

                if (tmp == 'A') {
                        // tmp가 A인 경우
                        if (pr[0].service_time == 0)
                // process A의 service time이 0인 경우
                                tmp = '\0';
                                        // tmp를 NULL로
                }
                else if (tmp == 'B') {
                        // tmp가 B인 경우
                        if (pr[1].service_time == 0)
                // process B의 두번째 service time이 0인 경우
                                tmp = '\0';
                                        // tmp를 NULL로
```

```
                }
                else if (tmp == 'C') {
                        // tmp가 C인 경우
                        if (pr[2].service_time == 0)
                // process C의 세번째 service time이 0인 경우
                                tmp = '\0';
                                        // tmp를 NULL로
                }
                else if (tmp == 'D') {
                        // tmp가 D인 경우
                        if (pr[3].service_time == 0)
                // process D의 네번째 service time이 0인 경우
                                tmp = '\0';
                                        // tmp를 NULL로
                }
                else if (tmp == 'E') {
                        // tmp가 E인 경우
                        if (pr[4].service_time == 0)
                // process E의 다섯번째 service time이 0인 경우
                                tmp = '\0';
                                        // tmp를 NULL로
                }

                for (int j = 0; j < count - 1; j++)
                // 노드 개수 - 1만큼 반복
                        q.proc_name[j] = q.proc_name[j + 1];
                q.proc_name[count - 1] = tmp;
        }
        else
                none = 0;
}

void queueCheck(int i, int x) {
                // 매개변수 i와 x
        zombie_q = 0;
                                // FCFS인 경우 x = 1 / RR인 경우 x = 0

        if (pr[0].arrival_time == i) {
                // process A의 도착시간이 i인 경우
                queueShare(pr[0], x);
```

```
                        // queueShare 함수 실행
                if (x == 0)
                                        // x = 0인 경우
                        zombie_q = 1;
                                // zoombie_q 변수를 1로 초기화
        }
        else if (pr[1].arrival_time == i) {
        // process B의 도착시간이 i인 경우
                queueShare(pr[1], x);
                if (x == 0)
                        zombie_q = 1;
        }
        else if (pr[2].arrival_time == i) {
        // process C의 도착시간이 i인 경우
                queueShare(pr[2], x);
                if (x == 0)
                        zombie_q = 1;
        }
        else if (pr[3].arrival_time == i) {
        // process D의 도착시간이 i인 경우
                queueShare(pr[3], x);
                if (x == 0)
                        zombie_q = 1;
        }
        else if (pr[4].arrival_time == i) {
        // process E의 도착시간이 i인 경우
                queueShare(pr[4], x);
                if (x == 0)
                        zombie_q = 1;
        }
}

void queueDelete() {
                        // 큐를 지우는 함수
        none = 1;
        q.proc_name[0] = q.proc_name[1];
        q.proc_name[1] = q.proc_name[2];
        q.proc_name[2] = q.proc_name[3];
        q.proc_name[3] = q.proc_name[4];
}
```

```
void servicetimeCount() {
	if (q.proc_name[0] == 'A') {
		// process가 A인 경우
		pr[0].service_time--;
			// A의 service time 감소

		if (pr[0].service_time == 0)
		// service time이 0인 경우
			queueDelete();
				// queue 삭제
	}
	else if (q.proc_name[0] == 'B') {
	// process가 B인 경우
		pr[1].service_time--;
			// B의 service time 감소

		if (pr[1].service_time == 0)
		// service time이 0인 경우
			queueDelete();
				// queue 삭제
	}
	else if (q.proc_name[0] == 'C') {
	// process가 C인 경우
		pr[2].service_time--;
			// C의 service time 감소

		if (pr[2].service_time == 0)
		// service time이 0인 경우
			queueDelete();
				// queue 삭제
	}
	else if (q.proc_name[0] == 'D') {
	// process가 D인 경우
		pr[3].service_time--;
			// D의 service time 감소

		if (pr[3].service_time == 0)
		// service time이 0인 경우
			queueDelete();
```

```
                                        // queue 삭제
        }
        else if (q.proc_name[0] == 'E') {
        // process가 E인 경우
                pr[4].service_time--;
                        // E의 service time 감소

                if (pr[4].service_time == 0)
                // service time이 0인 경우
                        queueDelete();
                                // queue 삭제
        }
}

void initQueue() {
        q1 = (LinkedQueue *)malloc(sizeof(Queue));
        // Queue(q1) 생성
        q1->count = 0;
        q1->front = NULL;
                                // q1 초기화
        q1->rear = NULL;

        q2 = (LinkedQueue *)malloc(sizeof(Queue));
        // Queue(q2) 생성
        q2->count = 0;
        q2->front = NULL;
                                // q2 초기화
        q2->rear = NULL;

        q3 = (LinkedQueue *)malloc(sizeof(Queue));
        // Queue(q3) 생성
        q3->count = 0;
        q3->front = NULL;
                                // q3 초기화
        q3->rear = NULL;

        q4 = (LinkedQueue *)malloc(sizeof(Queue));
        // Queue(q4) 생성
        q4->count = 0;
        q4->front = NULL;
```

```
                                // q4 초기화
        q4->rear = NULL;
}

void push_MLFQ(LinkedQueue *queue, char push) {
        // Queue Node 삽입 / push는 삽입할 문자
        ChainNode *node = (ChainNode *)malloc(sizeof(ChainNode));    //  삽입에
필요한 노드 생성
        node->data = push;
                                // 삽입 노드의 데이터를 push로 설정
        node->link = NULL;
                                // 삽입 노드의 link를 NULL로 설정

        if (queue->front == 0)
                        // 빈 큐인 경우
                queue->front = queue->rear = node;
                // front와 rear에 새로 만든 노드 삽입
        else {
                                        // 빈 큐가 아닐 경우
                queue->rear->link = node;
                        // queue 끝에 새로 만든 노드 삽입
                queue->rear = node;
                                // rear이 마지막 노드를 가리키도록
        }
        queue->count++;
                                        // 노드 개수 증가
}

char pop_MLFQ(LinkedQueue *queue) {
                // Queue Node 삭제
        char pop;
                                        // pop은 삭제할 문자
        ChainNode *node = (ChainNode *)malloc(sizeof(ChainNode));    //  삭제에
필요한 노드 생성

        if (queue->front == 0)
                        // 빈 큐인 경우 종료
                return 0;

        node = queue->front;
```

```
                        // queue의 맨 앞을 노드에 삽입
        pop = node->data;
                                // 노드의 데이터를 pop에 삽입
        queue->front = node->link;
                        // 삭제한 노드의 link를 queue의 맨 앞으로
        free(node);
                                        // 노드 반환
        queue->count--;
                                        // 노드 개수 감소


        return pop;
                                        // 삭제할 문자 리턴
}

int getRandom(int min, int max) {
                // 랜덤함수
        int num = max - min + 1;
        return min + (rand() % num);
}

void insert(char a[20][5], char ch, int i) {
// 매개변수 2차원 배열 a, 문자 ch, 시간 i
        if (ch == 'A')
                                // 2차원 배열에 삽입하는 insert 함수
                a[i][0] = 'A';
                                // MLFQ 함수에서 사용
        else if (ch == 'B')
                a[i][1] = 'B';
        else if (ch == 'C')
                a[i][2] = 'C';
        else if (ch == 'D')
                a[i][3] = 'D';
        else if (ch == 'E')
                a[i][4] = 'E';
}

void assign(char empty1[20][5], int i)
        // 매개변수 2차원 배열 empty1, 시간 i
{
                                                // FCFS, RR 함수에서 사용
```

```
        if (q.proc_name[0] == 'A')
                empty1[i][0] = 'A';
        else if (q.proc_name[0] == 'B')
                empty1[i][1] = 'B';
        else if (q.proc_name[0] == 'C')
                empty1[i][2] = 'C';
        else if (q.proc_name[0] == 'D')
                empty1[i][3] = 'D';
        else if (q.proc_name[0] == 'E')
                empty1[i][4] = 'E';

        servicetimeCount();
}

void print(char empty2[20][5])
                // 매개변수 2차원 배열 empty2를 이용한 출력함수
{
        for (int i = 0; i < 5; i++)
        {
                printf("%c : ", 65 + i);
                // A~E까지 출력

                for (int j = 0; j < 20; j++)
                {
                        if (empty2[j][i] == 'A')
                                printf("■");

                        else if (empty2[j][i] == 'B')
                                printf("■");

                        else if (empty2[j][i] == 'C')
                                printf("■");

                        else if (empty2[j][i] == 'D')
                                printf("■");

                        else if (empty2[j][i] == 'E')
                                printf("■");

                        else
```

```
                                printf("□");
                        }
                        printf("\n");
                }
                printf("\n");
}

void FCFS() {
        int x = 1;
                                                // RR과의 차이를 주기 위한 변수 x
        char FCFS[20][5] = { 0 };
                        // [20][5] 2차원 배열 초기화

        for (int i = 0; i < 20; i++) {
                queueCheck(i, x);
                assign(FCFS, i);
        }
        print(FCFS);
}

void RR() {
        int x = 0;
                                                // FCFS과의 차이를 주기 위한 변수 x
        char RR[20][5] = { 0 };

        for (int i = 0; i < 20; i++) {
                queueCheck(i, x);

                if (zombie_q == 0)
                                // zombie_q가 0인 경우
                        execute_RR();
                                // 다시 execute_RR 함수 실행
                assign(RR, i);
        }
        print(RR);
        printf("\n");
}

void MLFQ() {
                                // 변수 a는 시간의 흐름, c는 삭제한 문자가 무슨 문
```

```
자인지 파악
        int a = 0, c = 0, sum = 0;
                        // sum은 MLFQ에서 모든 큐의 노드 개수
        int b;
                                        // b는 구조체 변수 pr의 인덱스
        char ch;
                                        // ch는 pop 함수에서 삭제한 데이터
        char MLFQ[20][5];
                                // MLFQ 2차원 배열 정적 할당

        for (a = 0; a < 20; a++) {
                        // 시간의 흐름
                for (b = 0; b < 5; b++)
                        if (a == pr[b].arrival_time)
                // a와 도착 시간이 같은 process를 찾음
                                push_MLFQ(q1, pr[b].process);
                // q1에 삽입

                sum = q1->count + q2->count + q3->count + q4->count;

                                                // 아직 b가 들어오지 않은 상황에서
q1이 끝났기 때문에 q2로 가야하는 a를

                                                //  q1에  유지시키기  위해  모든
queue의 노드 개수를 카운트

                if (sum == 1) {
                                // queue 전체에 process가 하나 있는 경우
                        if (q4->count == 1) {
                        // q4 노드에 데이터가 있는 경우
                                ch = pop_MLFQ(q4);
                                // q4의 데이터를 삭제 후 ch에 삽입
                                push_MLFQ(q1, ch);
                                // 삭제한 데이터를 q1에 대입
                        }
                        else if (q3->count == 1) {
                        // q2 노드에 데이터가 있는 경우
                                ch = pop_MLFQ(q3);
                                // q3의 데이터를 삭제 후 ch에 삽입
                                push_MLFQ(q1, ch);
```

```
                        // 삽입한 데이터를 q1에 대입
                }
                else if (q2->count == 1) {
                // q1 노드에 데이터가 있는 경우
                        ch = pop_MLFQ(q2);
                        // q2의 데이터를 삭제 후 ch에 삽입
                        push_MLFQ(q1, ch);
                        // 삽입한 데이터를 q1에 대입
                }
        }

        if (q1->count != 0) {
                // q1 탐색
                ch = pop_MLFQ(q1);
                        // q1의 queue에서 제일 먼저 들어온 노드 삭제
                insert(MLFQ, ch, a);
                // 삽입

                for (c = 0; c < 5; c++)
                // 변수 c를 이용해 삭제한 문자가 무슨 문자인지 알아냄
                        if (ch == pr[c].process) {
                                pr[c].service_time--;
                // 해당 process의 service time 1 감소

                                if (pr[c].service_time != 0)
        // service time이 남아있는 경우
                                        push_MLFQ(q2, pr[c].process);
        // 다음 queueu에 삽입
                        }
        }
        else if (q2->count != 0) {
                // q2 탐색
                ch = pop_MLFQ(q2);
                        // q2의 queue에서 제일 먼저 들어온 노드 삭제
                insert(MLFQ, ch, a);
                // 삽입

                for (c = 0; c < 5; c++)
                        if (ch == pr[c].process) {
                                pr[c].service_time--;
```

```
                                        if (pr[c].service_time != 0)
                                                push_MLFQ(q3, pr[c].process);
                                }
                        }
                        else if (q3->count != 0) {
                                // q3 탐색
                                ch = pop_MLFQ(q3);
                                        // q3의 queue에서 제일 먼저 들어온 노드 삭제
                                insert(MLFQ, ch, a);
                                // 삽입

                                for (c = 0; c < 5; c++)
                                        if (ch == pr[c].process) {
                                                pr[c].service_time--;

                                                if (pr[c].service_time != 0)
                                                        push_MLFQ(q4, pr[c].process);
                                        }
                        }
                        else {
                                                // q4 탐색
                                ch = pop_MLFQ(q4);
                                        // q4의 queue에서 제일 먼저 들어온 노드 삭제
                                insert(MLFQ, ch, a);
                                // 삭제

                                for (c = 0; c < 5; c++)
                                        if (ch == pr[c].process) {
                                                pr[c].service_time--;

                                                if (pr[c].service_time != 0)
                                                        push_MLFQ(q4, pr[c].process);
                                        }
                        }
                }

                print(MLFQ);
                                        // 2차원 배열 MLFQ 출력
                printf("\n");
```

```
}

void LOTTERY() {
	int arr[20];
				// 랜덤함수 선언을 위한 배열
	int min, max, x, y;

	char LOTTERY[20][2];
			// LOTTERY 2차원 배열 정적 할당

	srand(time(NULL));
				// 난수 출력용 srand 함수
	printf("LOTTERY 최솟값, 최댓값 입력 : ");
	scanf_s("%d %d", &min, &max);
	printf("LOTTERY 비율 입력 : ");
	scanf_s("%d %d", &x, &y);
	printf("수: ");

	for (int i = 0; i < 20; i++) {
		arr[i] = getRandom(min, max);
		// 랜덤함수 getRandom 실행

		if (arr[i] < 10)
			// 출력되는 숫자들을 2글자로 맞추기 위함
			printf("0");
		printf("%d  ", arr[i]);
	}
	printf("\n");

	for (int j = 0; j < 2; j++) {
		// A와 B의 행
		printf("%c :", 65 + j);
			// ASCII code A, B 출력

		for (int i = 0; i < 20; i++) {
		// LOTTERY 20번 수행
			if (arr[i] <= x && j == 0)
			// LOTTERY A 출력
				LOTTERY[i][j] = printf(" ■ ");
			else if (arr[i] > x && j == 1)
```

```
                // LOTTERY B 출력
                        LOTTERY[i][j] = printf(" ■ ");
                else
                                // A의 값이나 B의 값이 비어있으면 빈칸 출
력
                        LOTTERY[i][j] = printf(" □ ");
                }
                printf("\n");
        }
}
```

**include/lab1_sched_types.h**
**(실습에 사용할 구조체 및 구현할 함수에 대한 헤더 파일)**

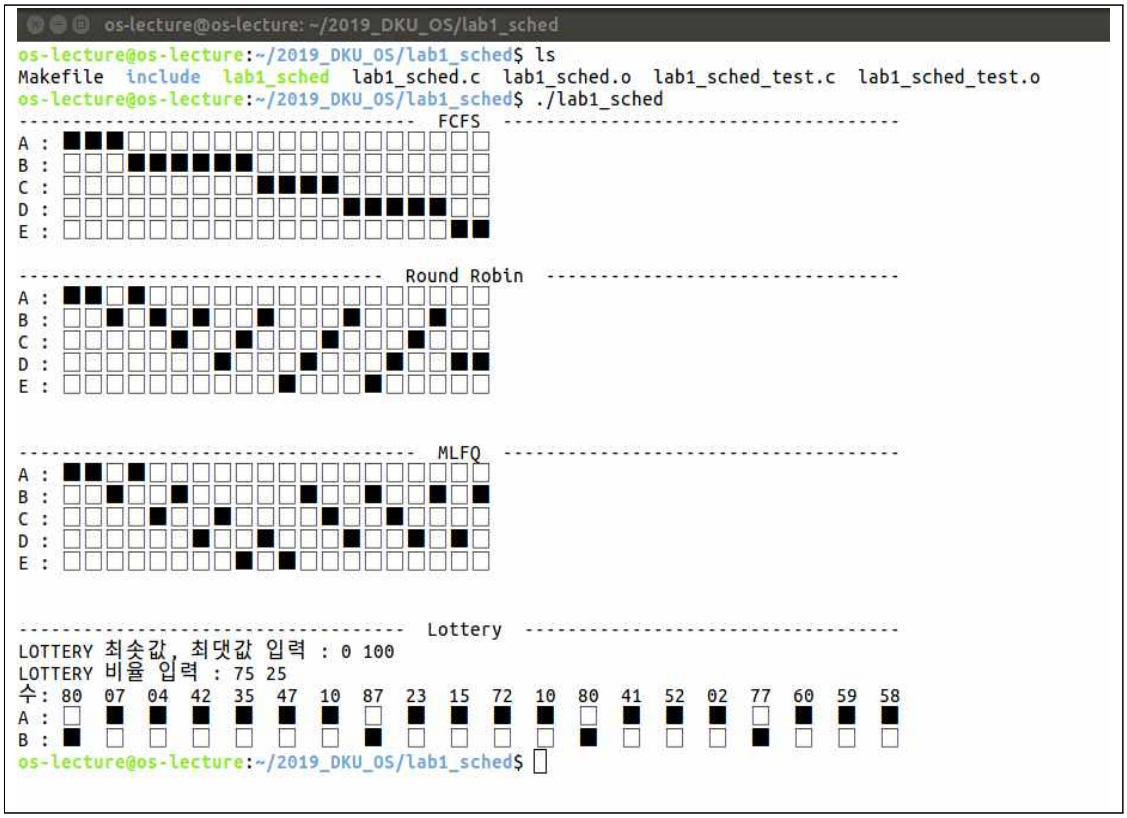
```
typedef struct process {
        char process;
        int arrival_time;
        int service_time;
}process;

typedef struct ChainNode {
        char data;
        struct ChainNode *link;
}ChainNode;

typedef struct Queue {
        char proc_name[6];
}Queue;

typedef struct LinkedQueue {
        ChainNode *front;
        ChainNode *rear;
        int count;
}LinkedQueue;
```

```
process pr[5];
Queue q;
LinkedQueue *q1;
LinkedQueue *q2;
LinkedQueue *q3;
LinkedQueue *q4;

void initWorkload();
void execute_RR();
void queueShare(struct process p, int x);
void queueCheck(int i, int x);
void queueDelete();
void servicetimeCount();
void initQueue();
void push_MLFQ(LinkedQueue *queue, char push);
char pop_MLFQ(LinkedQueue *queue);
int getRandom(int min, int max);
void insert(char a[20][5], char ch, int i);
void assign(char empty1[20][5], int i);
void print(char empty2[20][5]);
void FCFS();
void RR();
void MLFQ();
void LOTTERY();
```

**lab1_sched_test.c**
**(lab_sched.c에서 구현한 schedular algorithm을 수행할 test code)**

```
int main() {
        printf("------------------------------------                    FCFS
-----------------------------------\n");
        initWorkload();
                                // Workload 초기화
        FCFS();
                                        // FCFS 함수 실행

        printf("---------------------------------          Round       Robin
--------------------------------\n");
        initWorkload();
                                // Workload 초기화
        RR();
                                        // RR(Round Robin) 함수 실행

        printf("-------------------------------------                   MLFQ
-------------------------------------\n");
        initWorkload();
                                // Workload 초기화
        initQueue();
                                // Queue 초기화
        MLFQ();
                                        // MLFQ 함수 실행

        printf("-------------------------------------                   Lottery
-------------------------------------\n");
        LOTTERY();
                                        // LOTTERY 함수 실행

        return 0;
}
```

*2. 프로젝트 실행*


os-lecture@os-lecture: ~/2019_DKU_OS/lab1_sched
os-lecture@os-lecture:~/2019_DKU_OS/lab1_sched$ ls
Makefile include lab1_sched lab1_sched.c lab1_sched.o lab1_sched_test.c lab1_sched_test.o
os-lecture@os-lecture:~/2019_DKU_OS/lab1_sched$ ./lab1_sched
FCFS
A :
B :
C :
D :
E :
Round Robin
A :
B :
C :
D :
E :
MLFQ
A :
B :
C :
D :
E :
Lottery
LOTTERY 최솟값, 최댓값 입력 : 0 100
LOTTERY 비율 입력 : 75 25
수: 80 07 04 42 35 47 10 87 23 15 72 10 80 41 52 02 77 60 59 58
A :
B :
os-lecture@os-lecture:~/2019_DKU_OS/lab1_sched$

## Ⅲ. 프로젝트 결과

*2. 박민혁 고찰*

작년 2학기 시스템프로그래밍을 배울 때는, 팀 과제가 하나밖에 없었다. 사실 나는 팀 과제가 좋다. 왜냐하면 모르는 부분이 있으면 팀원에게 물어보면 되기 때문이다. 사실 저번 학기 시스템 프로그래밍 과제에선, 상민이가 대부분의 코딩을 맡아서 했다. 하지만, 이번 과제에선 상민이에게 더 도움이 될 수 있도록 노력을 많이 했다. 우선, 과제 기간이 길어서 다행이었다. 왜냐하면, 요즘 학생회 일이 바빠서 수업 집중을 잘 못하였고, 공부 할 시간도 없었으며, 수업을 빠지기도 했다. 그래서 운영체제를 최우선적으로 공부를 했다. 과제를 하려고, 3장인 scheduling 부분을 먼저 공부를 했지만, 아니나 다를까 이해가 하나도 되지 않았다. 그래서 1장부터 차분히 다시 공부를 하게 되었다. 그렇지만, 1장과 2장은 크게 도움 되지 않았고, 상민이에게 직접 물어봐서, 공부를 했고 일주일 동안 운영체제에 관한 것만 공부를 하게 되었다. 그리고 나서, 과제를 시작하였다. 사실 과제를 하면서, 구글링을 많이 했다. 혼자 생각하다가 막히는 부분이 많았다. 이론적으로나 그림을 그려서는 알겠는데 막상 코딩으로 짜려니 어디서부터 어떻게 시작해야 될지 몰라서, 틀을 대충 검색해서 정보를 얻었고, 그 틀을 이용해서 코딩을 짜기 시작했다. 우선 혼자서 코딩을 짜는데 오류가 많이 떴었다. 특히 MLFQ 부분에서 많이 막혔었다. 프로세스를 구성 하는 구조체들을 설계하는 것은 쉬웠다. 하지만 계속 빈 큐일 경우에, 계속적으로 프로그램이 끝나서, 오류가 많이 발생했었다. 예를 들어서, A가 빈 큐가 되면, 프로그램이 끝나 오류가 발생했다. 이러한 부분들을 큐 코딩을 잘 모르고 있어서, 그거 또한 공부를 하는데 오래 걸렸다. 그래서 상민이의 도움을 받았고, 상민이와 같이 날을 잡아서 10시간 정도 같이 과제를 했다. 처음 코딩을 짰을 때는 네모 칸이 아닌, A와 B의 형태로 나왔다. 하지만 보기 좋지 안 좋아서, 빈 네모 칸과 꽉 찬 네모 칸으로 바꾸었다. 처음 코딩을 완성 했을 때, 750줄 정도의 코딩이 나왔다. 하지만 필요 없는 부분들도 있어서 수정을 했다. 수정 부분은 상민이가 많이 했다. 상민이가 수정한 결과 Main 함수가 깔끔해졌고, 좀 더 가독성이 높아졌다. 한 300줄 정도 줄였다. 이번 과제를 하면서, 물론 과제도 중요하지만, 운영체제 공부를 하면서 조금 뒤 떨어졌던 부분들을 다시 따라갈 수 있어서 좋았다. 앞으로 팀 과제가 많이 남았지만, 상민이랑 계속적으로 좋은 결과를 만들 수 있을 것 같다. 비록 교수님께는 만족스러운 결과물이 아닐 수는 있지만, 그래도 나는 나름 결과가 정확히 딱 떠서 좋았다.

*1. 이상민 고찰*

개인적으로 이번 과제는 정말 어려웠다. 작년 시스템 프로그래밍 수업 때 했던 마지막 팀 과제는 아무것도 아니였구나라는 생각이 들었다. 우선 scheduling algorithm을 꼼꼼하게 공부했다. 과제에서 나온 알고리즘은 들어온 순서대로 처리를 해주는 FCFS, 이 FCFS에 preemptive가 추가된 Round Robin, 그리고 큐를 여러 개 사용해 우선순위를 주는 MLFQ이다.
개념을 이해하고 그것을 바탕으로 scheduling 그림을 그리는 것은 그다지 어렵지 않았는데, 막상 코딩을 하려고 하니 쉽지가 않았다. 리눅스는 C를 지원하기 때문에 C++에서처럼 class가 아닌 구조체를 사용해야 했다. 그리고 자료구조 수업 때 배운 연결리스트나 연결큐 같은 개념들을 사용했다. 정확한 코드가 기억이 나지 않아 작년에 사용했던 ppt를 참고한 것이 도움이 되었다.

박민혁 학생과 보너스 과제인 lottery를 먼저 짰다. lottery는 비교적 쉬운 편이었다. getRandom()이라는 함수 속에서 rand()를 사용해 난수를 리턴받아 사용했다. 일부러 입력을 받아 사용할 수 있게 함으로써 활용도를 높여보았다. 우선 ppt 3장에 있는 것처럼 workload를 주기 위해 process 구조체를 사용하였다. A부터 E까지의 문자와 arrival time, service time을 이용해 scheduling을 하도록 했다. initWorkload()라는 함수를 이용해 모든 process들을 초기화해주었다. 처음 코딩을 했을 때에는 현재 queueShare() 함수를 FCFS와 Round Robin에서 각각 한 번씩 사용했었다. 이 함수는 Queue 구조체 속 process name 배열이 비어있으면 queueCheck() 함수에서 매개변수로 받은 process를 저장해주는 함수이다. 두 개의 scheduling에서 동시에 사용할 수 있기 때문에 강제로 x라는 매개변수를 주었고 그 값이 0이면 Round Robin을, 1이면 FCFS를 실행하도록 코드를 짰다.
FCFS()와 RR() 함수에서는 assign()이라는 함수를 사용하였는데, Queue 구조체를 사용해 process 이름이 배열 각 값과 일치할 경우 2차원 배열 해당 위치에 넣어주었다. 이 2차원 배열은 print()라는 출력 함수에서도 사용하는데, 조금 더 보기 좋게 정렬하기 위해서 실행화면과 같이 네모 칸을 사용했다. 2차원 배열에 넣어줬다는 것은 process를 실행했다는 뜻이기 때문에 service time을 1씩 감소시켜 주었다.

가장 고생했던 것은 MLFQ를 구현하는 것이었다. 포인터를 사용하는 것이 익숙하지가 않아서 큰 어려움을 겪었다. 연결리스트와 연결큐를 공부하면서 사용했던 자료들을 보며 그림을 그려서 했는데도 쉽지가 않았다. 우리는 time quantum이 끝날 때마다 우선순위가 높은 큐를 찾아 삭제하고, 만약 삭제한 process의 service time이 남아있다면 그 다음 우선순위 큐로 집어넣은 후 이것을 time quantum이 끝날 때마다 반복하고 싶었다. 하지만 처음 process A만 실행되고 있을 때 A를 실행하고 q2로 내리는 것이 아니라 q1에서 머물게 하고 있어야 하는 점이 문제였다. 그래서 A를 애

초에 q2에 넣고 time quantum이 끝났는데도 q1이 비어있으면 우선순위를 올려주는 식으로 코드를 작성했다. 그렇게 함으로써 B가 들어오기 전 A의 우선순위가 내려가는 문제를 해결할 수 있었다. 이 과정에서 검색도 많이 했고 선배들에게 물어보기도 했다. 그 결과 큐에 들어있는 노드 개수를 사용하라는 조언을 받았다. 노드 개수를 센 뒤 모든 큐에 하나의 process만 들어있다면 그 process를 q1으로 올려주는 방법이었다. 혼자 힘으로 해결하지 못해서 아쉽긴 했지만 그래도 구현을 할 수 있게 되어 기뻤다.

팀 과제를 할 때마다 서로의 부족한 부분을 채워줄 수 있다는 점이 참 좋은 것 같다. 사실 C언어를 완전히 구사할 정도의 수준이 아니라고 생각했기 때문에 과제 시작 전에 두려움을 갖고 있었다. 포인터나 구조체를 사용하는 방법을 박민혁 학생이 잘 숙지하고 있었고, 많은 도움을 받을 수 있었다. 솔직히 이 과제가 혼자 하는 과제였다면 시작도 해보지 못한 채 포기했을 것 같다. 교수님께서도 그러한 점을 알고 계셨기 때문에 팀 과제로 내주셨다고 생각한다. 비록 여러 자료들을 사용해서 온전히 둘이서만, 완벽하게 구현한 프로젝트는 아니지만 상당히 많은 양의 코드를 조작하고 구현해보았다는 점이 뜻깊었다.