# REPORT


## 제목: 운영체제 과제#2

**과목명** : 운영체제
**담당교수**: 최종무 교수님
**이름**: 최지윤
**학번**: 32194747
**제출일**: 2020.04.02


단국대학교
Dankook University

## I. 프로젝트 분석 및 설명

### - FIFO Scheduling

FIFO(First In First Out)는 알고리즘에서 자주 등장하는 개념으로, 말 그대로 먼저 들어온 것이 먼저 나간다는 뜻이다. 따라서 FIFO Scheduling에서는 먼저 들어온 프로세스가 먼저 스케줄링되는 어찌보면 단순하고 기본적인 스케줄링 방법이다. 아직 실행이되지 않은 프로세스 중에서 arrival time이 가장 빠른 프로세스부터 각자의 service time만큼 할당받을 수 있도록 구현해보았다. FIFO는 그저 먼저 도착한 프로세스 순서대로만 수행시키면 되기 때문에 구현이 쉽고 간단하다. 하지만 만약 먼저 도착한 프로세스들 중에 아주 긴 작업이 있을 경우에는 다른 프로세스들이 그만큼 밀리게 되므로 비효율적이다.

### - SJF Scheduling

SJF 스케줄링은 가장 짧은 작업을 먼저 수행하는 방법이다. 이미 도착해있지만 아직 수행되지 않은 프로세스 중에서 service time이 가장 짧은 프로세스부터 각자의 service time 만큼 할당받을 수 있도록 구현하였다. 수행 시간이 짧은 프로세스부터 스케줄링하기 때문에 언뜻보면 FIFO의 단점인 프로세스 밀림 현상을 해결하는 것 같아보이지만, 만약 짧은 프로세스들이 하나의 긴 프로세스보다 약간이라도 늦게 도착했다면 기존의 긴 프로세스가 수행되느라고 다른 프로세스들이 또 대기해야하는 같은 문제가 발생할 수 있다.

### - RR Scheduling

RR(Round Robin) 스케줄링은 FIFO에 time quantom이라는 개념을 추가한 방법이다. 먼저 도착한 순서대로 스케줄링하되 time quantom만큼씩만 쪼개서 실행시킨다. 이때는 queue를 사용하게 되는데, time quantom만큼 실행을 마친 프로세스는 queue에서 나와 맨 뒤로 다시 들어간다. 이때, 새로운 프로세스가 들어오면 이를 알고 스케줄러는 실행되고있는 프로세스의 time quantom을 모두 사용했는지 확인한 후에 다음 프로세스로 CPU를 넘겨주는 과정이 중요하다. 따라서, 전반적인 FIFO 구조에서 조건문을 추가하여 time quantom만큼 실행되었는지 체크하면서 새로운 프로세스가 queue에 도착했는지 확인하는 과정을 중점적으로 구현하였다.

### - MLFQ Scheduling

MLFQ(Multi Level Feedback Queue) 스케줄링은 커널 내의 준비 큐를 여러개의 큐로 분리하여 queue 사이에도 우선순위를 부여하게 된다. 최상위 큐는 우선순위가 가장 높고 내려갈수록 우선순위가 낮아진다. 각 큐는 레벨에 따라 time quantom을 가지고 있으며 특정 규칙에 따라 우선순위를 높일지 낮출지 결정한다. MLFQ 스케줄링의 5가지 규칙은 다음과 같다.

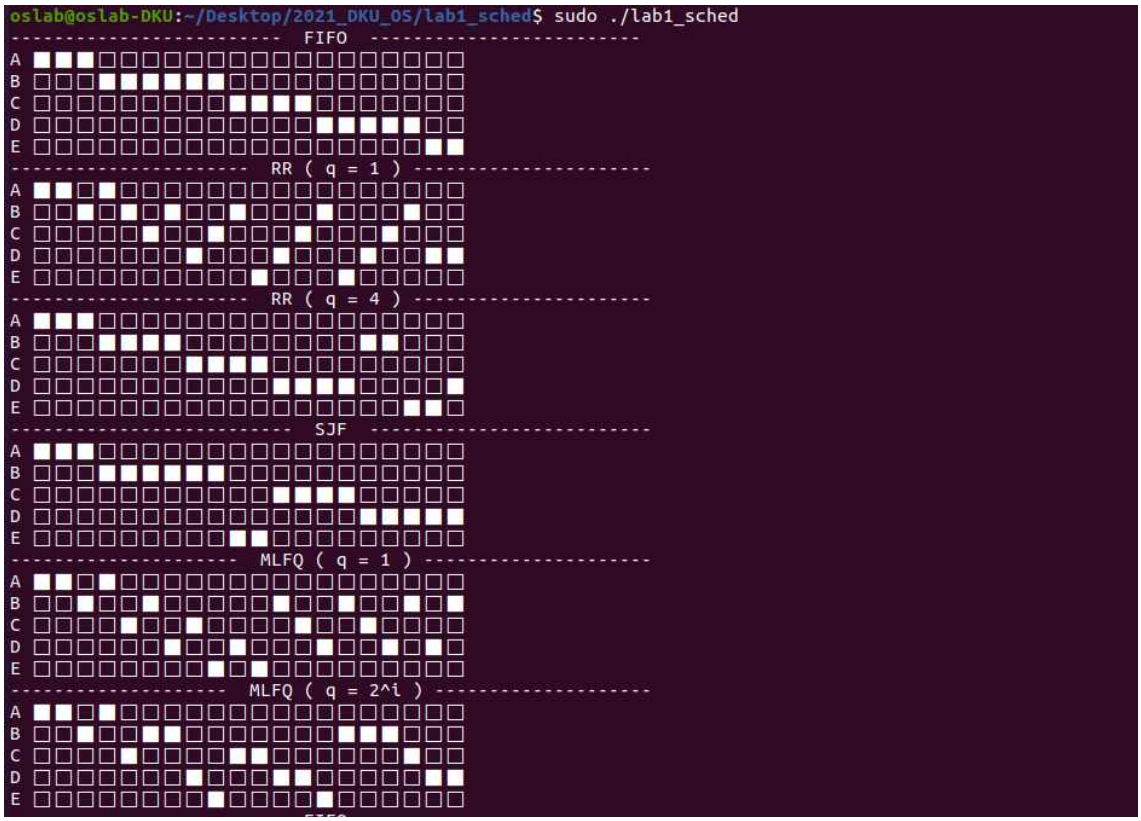
1. A의 우선순위가 B보다 높으면 A 실행
2. A와 B의 우선순위가 같으면 RR로 A와 B를 수행
3. 시스템에 프로세스가 처음 들어오면, 최상위 큐에 넣어줌
4. 프로세스가 주어진 레벨에 주어진 time_slice를 모두 사용했으면 우선순위를 낮춤
5. S라는 시간이 지나면 모든 프로세스를 최상위 큐로 끌어올림

구현은 큐가 여러 단계로 나뉘어 있다는 것에 초점을 맞추어 각 큐에 주어진 타임 퀀텀을 모두 사용하면 다음 큐로 보내는 방식에 집중하여 시도하였다.

## II. 실행 결과

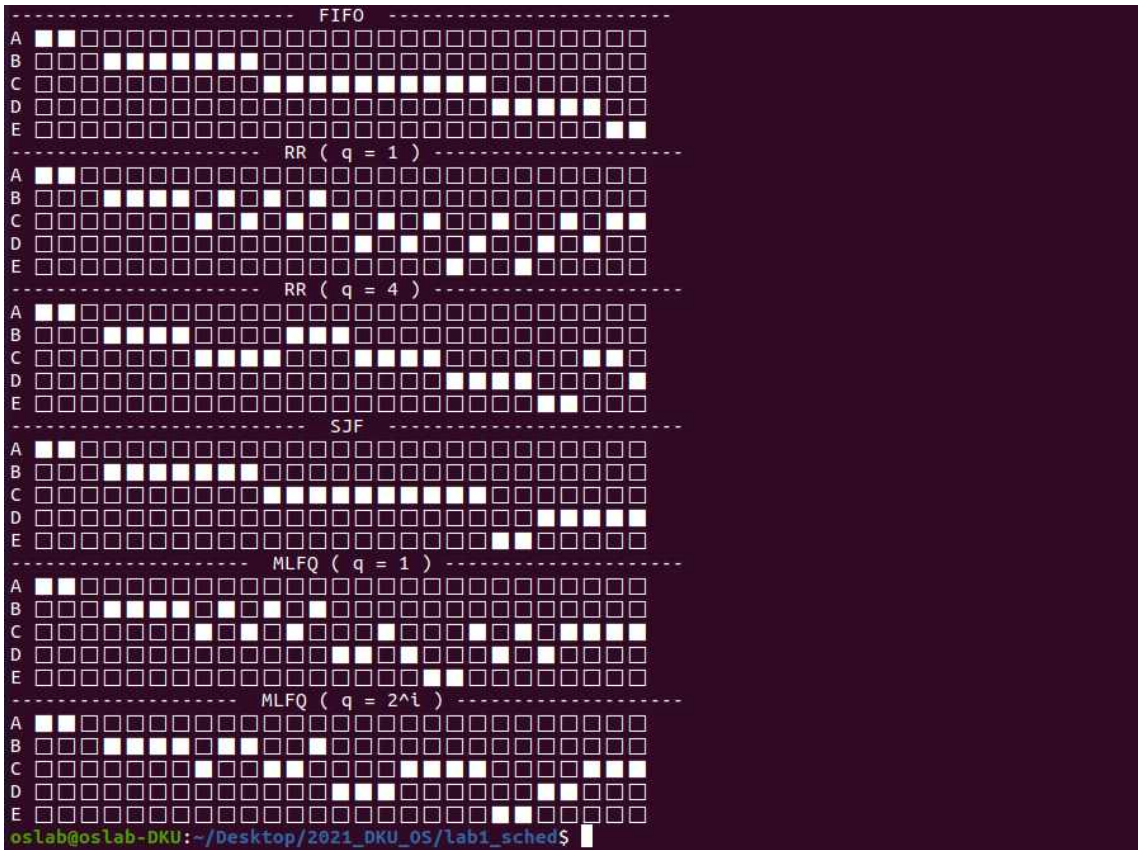
### ■ 24페이지 워크로드 수행 결과

| Process | Arrival Time | Service Time |
|---|---|---|
| A | 0 | 3 |
| B | 2 | 6 |
| C | 4 | 4 |
| D | 6 | 5 |
| E | 8 | 2 |



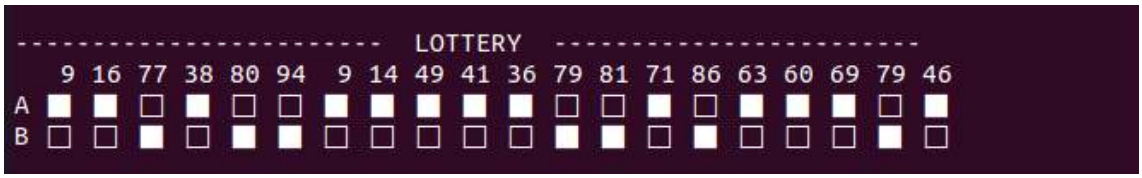
■ 새로운 워크로드 수행 결과

| Process | Arrival Time | Service Time |
|---|---|---|
| A | 0 | 2 |
| B | 3 | 7 |
| C | 7 | 10 |
| D | 13 | 5 |
| E | 17 | 2 |

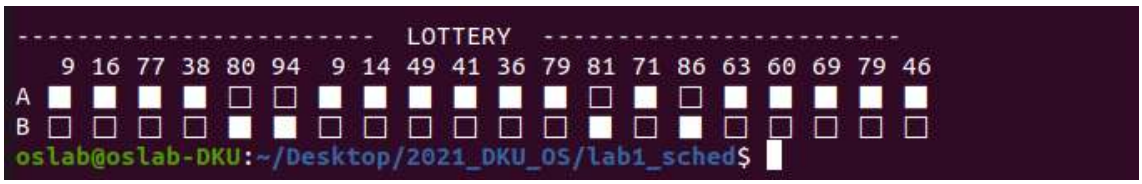


## III. 보너스 문제 ( Lottery Scheduling )

■ total: 0~99 // A ticket: 75  // B ticket: 25 일 때 수행결과




■ total: 0~99 // A ticket: 80  // B ticket: 20 일 때 수행결과




## IV. 고찰

시스템프로그래밍 수업을 들을 때 시프 과제보다 더 어려운 과제가 있을까 싶었는데 바로 여기에 있었다. 운영체제 과제를 하다보니 '시프과제는 정말 할만했구나' 싶었다. 그만큼 어렵기도 어렵고 시행착오가 아주 많았던 과제였다. 맨 처음 스케줄링 구현이라는 과제를 받고 각각의 스케줄링 알고리즘의 개념과 원리를 완벽하게 익히고 나니까 대충 머릿속에 크게 그림이 그려졌다. 그래서 자신감도 생겼고 금방 구현할 수 있을 것 같았다. 하지만 막상 코드를 짜려고 보니까 현실의 벽처럼 막막하게 느껴졌다. FIFO와 SJF는 그나마 알고리즘이 간단하여 어찌저찌하다보면 뭔가 완성이 되어가는 느낌이 들었지만 RR이나 MLFQ와 같이 조금 복잡한 알고리즘의 경우 연결리스트나 연결 큐와같은 개념들까지 도입되어 더더욱 어렵고 막막하게 느껴졌다. 그렇다보니 이번 과제의 경우 특히나 구글의 힘을 많이 빌리게 되었다. 코드 작성하는 데에 있어 구글에서 참고한 부분이 꽤나 많다. 하지만 구글링도 실력이라고 생각하기 때문에 검색을 해서라도 끝까지 구현해낸 내 자신이 대견하다.

과제에 출력 형식을 어떤식으로 해야할지 명시가 되어있지 않아서 어떻게 결과를 보여줄지 고민하다가 그냥 24페이지의 그림과 같이 한눈에 보여줬음 좋겠다는 생각하던 와중에 동기가 스케줄링이되지 않으면 빈 네모, 시케줄링이 되면 꽉 찬 네모를 출력하면 좋을 것 같다는 아이디어를 공유해주어 그렇게 하기로 했다. 또, 이번에 새로 깐 virtual box와 우분투 환경이 익숙하지 않아 비쥬얼 스튜디오에 모든 코드를 일단 작성하고 완성이도면 이를 복사하여 우분투 환경에서 실행시켜보는 방식으로 진

행했다. 하지만 나름대로 코드를 짜본 후 기대감을 안고 virtual box를 통해 코드를 실행시켜보려고 했는데 vi 편집기부터 작동하지 않고 자꾸만 접근권한 에러가 났다. 원인을 해결하려면 너무 많은 시간과 노력이 들 것 같아서 그냥 sudo를 앞에 붙혀 관리자 권한으로 실행시켰다.

FIFO를 가장 먼저 구현해보았다. 들어오는 Task 순서대로만 먼저 처리해주면 되어서 코드를 짜는데 크게 막히는 부분이 없었다. 하지만 다 짜고서 실행을 시켜보니 계속해서 오류가 발생했다. Task와 Task 사이에 공백이 생기는 경우를 생각 못했다. 그래서 부랴부랴 이전 Task의 arrival time과 service time을 더해서 현재 Task의 arrival time보다 작으면 time_count를 하나씩 늘려주고 더한 값이 현재 Task의 arrival time과 같을 때까지 빈 네모칸을 출력하면 하는 식으로 해결했다.
FIFO를 구현하고 나니 SJF는 너무 쉽게 코드를 완성 시켰다. 그냥 FIFO 코드를 복사하여 이름만 SJF로 바꿔주고, arrival time을 비교해주는 조건문을 이미 도착해있는 프로세스 중에서 service time을 비교해주는 조건문으로 그 한줄만 바꾸어주어 구현했다. 난관은 이제 RR 구현을 시작하면서 시작되었다. 가장 큰 문제는 아까처럼 프로세스들의 도착시간 사이에 빈 공간이 생겼을 때 무한루프에 빠진다는 것이었다. 그래서 큐가 비어진 상황에서 도착 시간을 검사하는 로직을 수행하여 큐에 새로운 프로세스를 넣고 반복문의 범위를 수정하게 되면서 겨우 에러처리를 하였다. 이렇게 힘들게 RR을 구현하고 보니 더 복잡한 MLFQ가 기다리고 있었다. MLFQ를 짜면서 포기하고 싶은 생각을 정말 많이 했다. 만약에 정 안되면 MLFQ 빼고 제출해야겠다는 마인드로 마음을 비웠다. 최대한 MLFQ 규칙을 곱씹으면서 단계적으로 구현하려고 애썼다. 여러 개의 큐가 우선순위를 다 가지고있고, 고려해야할 부분이 많다보니까 크고 작은 에러들이 많이 발생했다. 최상위 큐가 완전 비어버리고 종료되는 경우, 에러는 없는데 결과가 다르게 나오는 경우, 등등 많은 시행착오가 있었지만 그럴 때마다 일일이 에러핸들링을 해주어 결국에 오류없이 원하는 결과가 나오는 것을 보고 진심으로 환호했다. 원래는 보너스 문제까진 진짜 상상도 못했는데 과제 보고서를 작성하다가 로또도 한번 해볼까?하는 생각이 들어 즉석에서 코드를 짜봤다. Lottery도 ppt에 나와있는 결과물을 바탕으로 출력되게 만들었는데, 위의 모든 스케줄링 알고리즘에 비하면 구현이 너무나도 쉬웠다. 쉬운건지 아님 이전 구현들이 너무 어려웠어서 쉽게 느껴진 건지는 잘 모르겠다. 그냥 랜덤함수로 난수를 생성하고 티켓 범위에 따라 스케줄링을 하면 되는 아주 간단한 작업이라 금방 끝낼 수 있었다.

처음에는 윈도우가 익숙한 나머지 virtual machine을 이용해 우분투 환경에서 작업하는 것이 너무 불편했다. 하지만 코드 수행을 반복하면서 노가다를 많이하다보니 많이 익숙해지고 조작도 꽤 빨라졌다. 과제를 늦게 시작한 동기가 gcc로 컴파일하는데 왜 오류가 나는건지 봐달라고 하길래 “그거 그냥 make 쓰면 알아서 컴파일 돼”라고 말해줬더니 그 동기가 굉장히 놀라워하는 모습을 보자 대단한 것은 아니지만 왠지

모르게 뿌듯하기도 했다. 사실은 나도 처음엔 gcc -o로 컴파일 하려다가 시프 때 배운 make 명령어가 생각나서 써먹었은 것이었다. 너무 편리했다. 비록 머리를 쥐어짤 정도로 어려웠던 과제였지만 리눅스 환경에도 많이 익숙해지고, 스케줄링에 대해서도 평생 까먹지 못할 만큼 이해한 것에 대해서 큰 도움이 된 것 같다. 정말 값진 결과물이었다.