Operating System
Lab 1

단국대학교
Dankook University

이름 : 박동학, 홍승기
학번 : 32151648, 32155068
단국대학교 소프트웨어학과

목차

I. 분석

1. 프로젝트 분석

이번 프로젝트는 Operating System(O.S)에서 프로그램을 수행할 때 원활하게, 모든 프로그램이 각자의 CPU를 가지고 있는 것처럼 만들어 주는 Scheduling을 하는 Scheduler를 구현하는 것이다.

Scheduling

Scheduling은 위에서 말했듯 간단히 말해서 디스크에 있는 Program이 수행 돼서 process 상태가 되었을 때 CPU를 사용하는데, 우리는 process들이 모두 각자의 CPU 위에서 동작 돼서 연속적으로 수행되는 것처럼 사용자가 느끼게하기 위한 CPU 할당을 이야기한다.

하지만 실제로 시스템 내부에서는 Scheduler가 바쁘게 context switch를 수행하면서 실행되고 있는 process들에게 연속적으로 CPU를 할당시키고 옮기는 과정을 반복하고 있다. 우리가 실제 사용하는 환경에서는 정말 체감 할 수 없을 정도로 짧은 시간 안에 모든 일이 일어나서, 실제 사용하는 사용자 입장에서는 이 과정을 체감 하기란 사실상 불가능 할 수 있다고 볼 수 있다. 스케줄링 방식을 구분하는 방법은 많지만 우리는 비선점형과 선점형에 대해서 알아볼 것이다.

1. 비선점형

비선점형 스케줄링은 어떤 process가 CPU를 할당 받으면 그 process가 종료 되거나 입출력 요구가 발생 되어서 자발적으로 중지 될 때까지 실행되도록 보장 받는 것을 말한다. 순서대로 (도착시간대로) 처리되는 공정성이 있고 다음에 처리해야 할 프로세스와 관계없이 응답 시간을 예상할 수 있고, context switch에 의한 Overhead가 선점 방식 보다 적다. 일괄처리 시스템(Batch System)에서 사용하는 것이 적합함, CPU 사용 시간이 긴 프로세스가 실행중이면 CPU사용이 적은 프로세스들을 오래 기다리게 할 수 있어서 처리율이 떨어짐

2. 선점형

선점형 스케줄링 방식은 어떤 process 가 CPU 를 할당받아 실행하고 있는 상황에서 다른 process 예를 들어 Service Time 이 더 적은 process 가 들어오면 실행 중인 process 를 중지시키고 CPU 를 강제로 점유하게 하는 것이다. 빠른 응답시간이 중요한 대화식 시스템(Interactive System)에서 사용하는 것이 적합함.

성능 분석 지표

1. Turn Around Time

- CPU 에 작업 요청이 있고 나서 얼마나 지난 후에 작업이 완료 되는가에 대한 지표이다.

2. Response Time

- CPU 에 작업 요청 후 얼마나 지난 후에 처음으로 CPU 가 작업을 수행하는 지에 대한 지표이다. 이는 주로 Real-Time 에 이용된다.

III. 프로젝트

1. 프로젝트 설계

이번 프로젝트에서 우리가 설계해볼 스케줄링 알고리즘으로는 크게 세 가지가 있다.

1. FCFS(First Come First Served)

2. RR(Round Robin)

3. MLFQ(Multi Level Feedback Queue)

4. Lottery

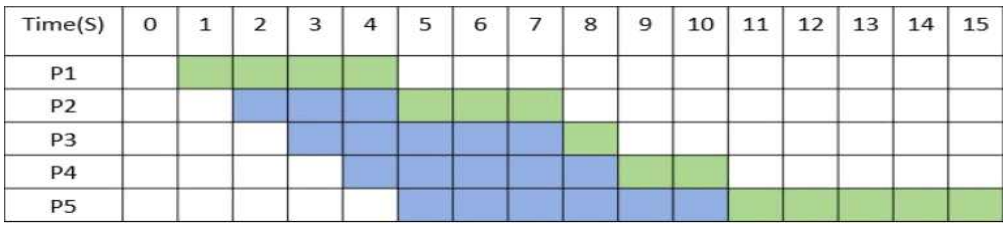
0. Queue

우선적으로 프로젝트를 수행하기 전에 Queue를 구현했다. Queue를 구현하는 것은 연결리스트를 이용하여 구현하였으며 이전에 구현해 놓은 소스코드를 재사용하였다.

1. FCFS

가장 기본적인 스케줄링 알고리즘이다. Queue 를 사용해서 Queue 의 기본적인 특성인 FIFO(First in Frist out)를 이용한 것, 쉽게 말하면 선착순으로 스케줄링 하겠다는 알고리즘 이다.

우리가 설계를 기획하면서 1 차적으로 필요하다고 생각한 것은 큐 한 개와 process 의 정보가 들어있는 구조체이다. 구조체에 process 의 이름, 도착시간, 서비스 타임 등을 넣어두고 큐에 도착순서대로 넣고 서비스 타임이 끝날 경우 큐에서 삭제하는 식으로 구현을 시도 해보기로 했다.

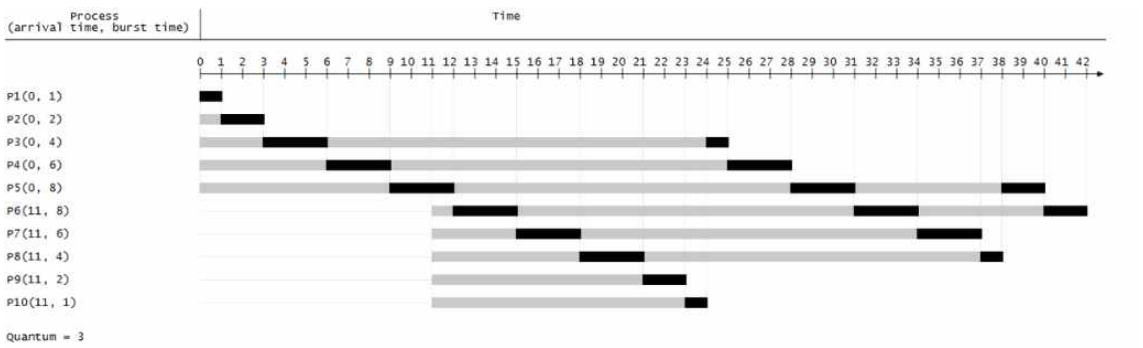


FCFS(우리가 기대하는 실행결과는 위 그림의 모습과 같음)

2. RR (Round Robin)

Round robin 스케줄링 알고리즘은 기본적으로 FCFS 에 time_slice 라는 것을 추가해서 만약에 한 process 가 서비스 타임이 3 인데 time_slice 가 1 이면 time_slice 만큼 수행하고 준비 큐에 도착해 있는 다른 process 를 스케줄링 해주는 알고리즘이다. 여기서 time_slice 를 다 사용한 process 는 큐에서 나와 큐의 맨 뒤로 들어가게 됨 (Enqueue 와 Dequeue 가 같이 일어남) 여기서 우리가 제대로 알고 넘어가야 할 부분은 새로운 process 가 들어오면 그것을 알고 스케줄러가 실행되고 있는 process 가 time_slice 를 다 사용했는지 확인하여 다음 process 로 CPU 점유를 넘겨주는 과정을 제대로 구현해야 한다는 것이다. 이 부분이 RR 스케줄링의 핵심 이라고 볼 수 있다. 이러한 점을 preempt 라고 한다.

우리가 RR 설계를 계획 할 때 넣어야 한다고 생각한 것은 기본적으로 FCFS 구조에 조건문을 넣어 time_slice 사용 여부를 체크하면서 새로운 process 가 큐에 도착했는지 확인을 하는 것을 기본적인 알고리즘이라 가정하고 설계를 시도하기로 했다.



(RR 수행시 예상 그림)

3. MLFQ

MLFQ 는 이름 뜻을 해석해보면 어떤 구조로 되어있는지 알수 있다, 이름 그대로 Multi level queue 를 사용하는데 feedback 기능이 있는 것이다. 쉽게 말해서 큐가 여러 단계로 나뉘어 있다. 각각의 큐는 우선순위가 존재하는데 우리가 설계해볼 MLFQ 는 3 단계로 나누어서 만들어 볼 것이다. 최상위 큐는 우선순위가 가장 높고 밑으로 내려갈수록 우선순위가 낮아지게 된다. 우선순위가 높을수록 먼저 실행되게 되고, 우선순위가 낮은 큐에 들어있는 process 는 상위 큐에 process 가 존재하면 CPU 점유가 불가하다. 이름에 들어있는 Feedback 기능은 process 의 행동을 토대로 우선순위를 낮출지 높일지 결정하는 것을 의미한다. 예를 들어 MLFQ 는 기본적으로 RR 의 방식과 유사하다. 각각 큐는 레벨에 따라서 time_slice 를 가지고 있는데, 만약

최상위 큐의 time_slice 가 1 이고 다음이 2, 그 다음 레벨이 4 라고 했을 때, 최상위 큐에 들어있던 어떤 process 가 서비스타임이 3 이여서 time_slice 1 만큼을 다 사용 하고 나면, 다음 큐로 우선순위를 낮추는 과정이 일어나게 된다.

밑의 그림은 MLFQ 의 규칙을 나타낸다,

- **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 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 5:** After some time period *S*, move all the jobs in the system to the topmost queue.

규칙 1. A 의 우선순위가 B 보다 높으면 A 실행

규칙 2. A 와 B 의 우선순위가 같으면 RR 로 A 와 B 를 수행

규칙 3. 시스템에 process 가 처음 들어오면, 최상위 큐에 넣어줌

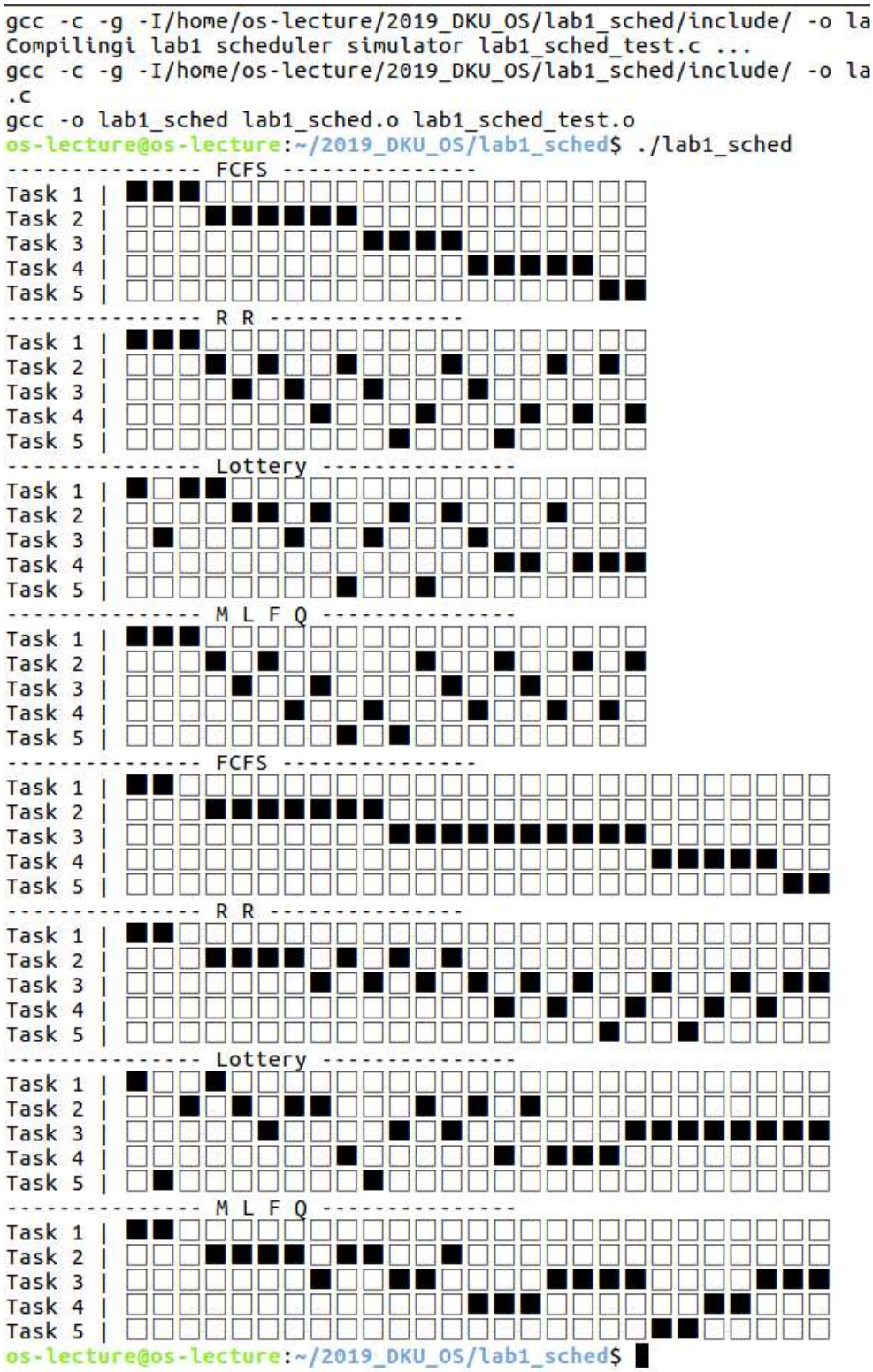
규칙 4. process 가 주어진 level 에 주어진 time_slice 를 다 사용했으면 우선순위를 낮춤

규칙 5. S 라는 시간이 지나면 모든 프로세스를 최상위 큐로 부스트함

이렇게 다섯 가지 규칙이 존재하는데, 우리가 설계할 MLFQ 는 규칙 5 를 신경 쓰지 않고 구현하는 것이 목표이다. 기본적으로 RR 과 많이 유사하지만, 큐가 여러 단계로 나뉘어 있다는 것에 초점을 맞춰서 각 큐에 주어진 time_slice 를 다 사용하면 다음 큐로 보내는 방식으로 구현을 시도해 볼 것이다.

IV. 논의

1. 수행 결과


gcc -c -g -I/home/os-lecture/2019_DKU_OS/lab1_sched/include/ -o la
Compilingi lab1 scheduler simulator lab1_sched_test.c ...
gcc -c -g -I/home/os-lecture/2019_DKU_OS/lab1_sched/include/ -o la
.c
gcc -o lab1_sched lab1_sched.o lab1_sched_test.o
os-lecture@os-lecture:~/2019_DKU_OS/lab1_sched$ ./lab1_sched
FCFS
Task 1 |
Task 2 |
Task 3 |
Task 4 |
Task 5 |
R R
Task 1 |
Task 2 |
Task 3 |
Task 4 |
Task 5 |
Lottery
Task 1 |
Task 2 |
Task 3 |
Task 4 |
Task 5 |
M L F Q
Task 1 |
Task 2 |
Task 3 |
Task 4 |
Task 5 |
FCFS
Task 1 |
Task 2 |
Task 3 |
Task 4 |
Task 5 |
R R
Task 1 |
Task 2 |
Task 3 |
Task 4 |
Task 5 |
Lottery
Task 1 |
Task 2 |
Task 3 |
Task 4 |
Task 5 |
M L F Q
Task 1 |
Task 2 |
Task 3 |
Task 4 |
Task 5 |
os-lecture@os-lecture:~/2019_DKU_OS/lab1_sched$

Case 1.

Task	도착시간	수행시간
1	0	3
2	3	6
3	9	4
4	13	5
5	18	2

Case 2.

Task	도착시간	수행시간
1	0	2
2	3	7
3	10	10
4	20	5
5	25	2

-> 예제는 수업자료에 나온 대로 1 개 여러 가지 경우가 나오도록 설정하여 1 개를 실행 해보았다.

프로젝트를 하면서 어려웠던 점

32151648 박동학

Lab #1 은 운영체제에서 핵심적인 기능을 담당하는 Scheduler 를 여러 기법을 통해서 구현해 보는 것이다. (정확하게는 시뮬레이션)

처음에 수업을 듣고 과제에 대한 설명을 들었을 때는 간단하게 구현할 수 있을 거라고 생각했다. 실제 시간에 맞추어 프로세스를 만드는 것도 아니고 우리가 시간을 주면서 구현하는 것이기 때문에 어렵지 않게 수행 할 수 있을 거라고 생각했다.

과제를 진행하면서 느낀 어려움은 아래와 같다.

1. 큐를 이용한 FCFS 에서의 수행시간에 공백 만들기

FCFS 의 원리는 First In First Out 이였기 때문에 처음 구현하는데 큰 어려움은 없었다. 들어오는 순서대로 큐에 넣고 서비스 타임이 끝나면 빼주는 형식으로 작성을 하고 테스트를 하는 부분에서도 지장이 없었다.

팀원과 다음 스케줄링으로 넘어가려고 했는데

'A 가 2 초에 끝나고 큐가 빈 상태에서 4 초에 B 가 도착하면 우리 프로그램은 정상으로 작동하는지가 궁금했다.'

반복문 안에서 Time_Slice 만큼 카운트를 해서 도착시간와 일치하면 Enqueue 를 하는 형식이었고 큐가 완전히 비게 되면 종료하는 형식이였기 때문에 A 가 끝나면 실행이 종료되어 버렸다. 해결을 위해서 여러 가지 발생 가능한 상황을 분석하여 조건문과 반복문으로 해결을 했다.

2. RR(Round Robin)에서의 큐 삽입 순서 정하기

RR 을 구현하는데 있어서는 우리가 수업시간에 배운 것을 토대로 작성하려고 했다.

처음 들어온 프로세스부터 수행하고 중간에 새로운 프로세스가 들어오면 큐에 넣고 타임 슬라이스가 끝나면 넘어가도록 설계를 진행하였지만 계속해서 순서가 다르게 나오는 문제가 발생하였다.

여러 시도 끝에 큐에 들어가는 순서와 도착하는 순서가 중요하다는 것을 알게 되었고 도착시간과 큐에 들어가는 순서를 맞춤으로 해결하였다.

3. MLFQ 에서 각 큐에 담긴 프로세스가 제대로 된 순서로 동작하게 하기

MLFQ 는 기본적으로 5 가지 규칙이 있지만 이번 과제에서 부스트는 실행하지 않았기 때문에 우리가 고려할 사항은 4 가지였다.

첫 번째와 두 번째는 우선순위를 고려하는 것으로서 우선순위를 큐를 이용해서 줬다. 따로 우선순위를 변수로 선언하지는 않고 높은 순서의 큐가 비었을 경우에만 다음 큐에 있는 프로세스가 수행 되도록 설계하였다.

하지만 MLFQ 에서는 이러한 사항들을 고려하기 위해서는 다양한 조건을 검사해 줘야 했다. 예를 들면 1 번 큐에서 어떤 경우에는 다시 1 번큐로 돌아가는지 또는 1 번큐가 지나고 2 번큐가 수행하는 사이에 새로운 프로세서가 들어오는 경우 등 많은 경우들이 있었고 여러 가지 테스트를 진행하면서 빠진 부분들을 채워주고 다시 알고리즘을 검사하는 등 오랜 시간에 걸려서 수정을 하면서 해결하였다.

사실 모든 부분에서 어려운 점이 하나씩 있었다. 심지어 그림을 그리는 것과 우분투에서 파일을 나누어 Make 하는 것 까지 다 익숙하지 않아 많은 시간이 걸렸던 것 같다.

처음에는 팀원이 FCFS 를 하고 내가 RR 을 하는 식으로 한 개씩 구현하려고 했지만 구현을 하다 보니 RR 은 FCFS 에 선점형을 추가한 형태이고 MLFQ 는 RR 을 여러 Level 로 나누어서 구현한다는 것을 알게 되었다. 그래서 스터디룸이나 여러 장소를 빌려가면서 칠판에 그리기도 하고 손으로 써보기도하고 정말 고생하면서 하나하나 다 직접 작성했던 과제인 만큼 시간은 많이 들었지만 큰 보람을 느낀 과제이다.

32155068 홍승기

이번 팀 프로젝트는 스케줄러의 종류인 FCFS, RR, MLFQ 를 구현하는 것 이였다.

FCFS 는 일반적인 큐를 사용한 스케줄러인데 연결 큐를 사용해서 구현을 시도해서 비교적 쉽게 구현 할 수 있을 줄 알았는데, 처음 생각했던 방식으로 구현을 시도했는데 계속 해서 오류가 발생해서 생각보다 시간이 오래 걸렸다. 처음 단순하게 생각했던 방식은 큐를 하나 만들고, 그 큐에 프로세스가 도착하는 순서대로 넣고, 실행을 시키는데 실행하면서 서비스타임이 0 이 되면 큐에서 빼서 삭제하는 원리인데, 여기서 고려 하지 않았던 부분이 만약 프로세스들의 도착시간에 빈공간이 생기게 되면 스케줄링하는 반복문이 종료되고 다음 프로세스를 못 넣는 문제가 계속 생겨서 이 문제를 해결하느라 거의 8 시간을 쓴 것 같다. 저 부분을 수정하고 나서는 별로 어렵지 않게 구현을 마쳤다.

두 번째 RR 은 FCFS 보다 더 알고리즘이 복잡해서 MLFQ 다음으로 신경을 많이 썼다. Time slice 가 끝나면 실행되고 있는 프로세스를 큐 맨 뒤에 넣고 삭제를 해야 하는데 FCFS 와 마찬가지로 프로세스들의 도착시간 사이에 빈공간이 생겼을 경우 이것 또한 오류가 생겨서 아예 무한루프에 빠지는 문제가 생겼었다. 그래서 생각해낸 방법이 변수하나를 미리 생성해놔서 빈 큐가 되는 상황에 도착시간 검사를 다시 실행해서 큐에 새로운 프로세스를 넣는 방식을 쓰고, 반복문 자체를 빈 큐일 때 까지 돌리지 않고 프로세스들의 서비스타임이 0 이 될 때까지 반복하는 방법으로 해결을 시도 했다. 이 방법을 실제 적용해서 시험해보는 시간만 10 시간 넘게 걸렸던 것 같다. 생각으로는 쉽게 적용되고 금방 해결 할 줄 알았는데, 조건문을 추가해야 되는 일이 많이 발생하다 보니, 코드에 순서가 약간씩만 바뀌어도 결과 자체가 완전히 달라지기도 하고, 코드가 길어지면 길어질수록 가독성이 안 좋아져서 프로그램을 짜는 나조차 수정하기 위해 문제가 되는 부분을 찾는데 시간이 너무 오래 걸렸다.

마지막 세 번째 MLFQ 는 정말 너무 어려웠다. 수업시간에 배운 배경지식을 바탕으로 RR 소스에 큐를 단계별로 만들어서 Time slice 가 끝나면 다음 큐로 내려주고 새로운 프로세스가 들어오면 최상위 큐에 넣는 방식으로 구현을 할려고 시도 했었는데, RR 과 유사한 부분도 물론 있지만 차이점이 존재해서 약간의 수정 작업이 좀 필요했었는데, 저 문제를 해결하고 프로그램을 도착시간을 바꿔서 실행 시켜보니, 이것도 앞에 스케줄러들과 똑같은 문제를 가지고 있었다. 문제가 된 원인은 RR 은 큐 하나로 구현해놨고 MLFQ 는 큐 여러 개에 우선순위가 있어서 다음 레벨 큐로 내려가게 되면 도착시간 사이에 차이가 있으면 최상위 큐가 빈 큐인 채로 종료된다는 걸 고려하지 못한 채로 코드를 짜고 실행을 시도해서

프로그램이 그냥 튕기는 것 이였다, 해결 방법으로 변수를 하나 미리 생성해 둬서 그 변수를 가지고 최상위 큐에 프로세스가 들어있지 않고 시간 안에 도착한 프로세스가 없으면 다음 도착시간을 검사하는 방식으로 해결을 시도했다. 그런데도 오류가 생겼었는데 오류의 원인이 A 라는 프로세스가 2 번큐로 내려간 상황에서 새로운 프로세스 B 가 최상위 큐에 들어왔을 때 최상위 큐로 들어가고 최상위 큐에 있는 B 를 2 번째 큐에 있는 A 의 TIME SLICE 만큼 실행하고 최상위 큐로 돌아가서 B 를 수행해야 되는데 이게 해결되지 않아서 생기는 문제 였다. 이 문제는 비교적 쉽게 2 번 큐에서도 프로세스가 들어오는지 확인해서 새로운 프로세스가 들어오면 break 를 해주는 것으로 해결했다.

보너스로 Lottery 스케줄러를 만들었는데 교수님이 다른 것보다 비교적 쉬울 거라고 하셔서 쉬울 줄 알았는데 생각보다 어려워서 정말 당황했다. 체감 상 FCFS 보다 어렵고 MLFQ 보다 약간 쉬운 정도? 라고 코드를 짜면서 느꼈는데, 큐를 사용하지 않아서 서비스타임이 끝나면 다음 걸로 지정해주고 티켓이 그 프로세스에 할당 되는지 검사 하는 게 오류가 진짜 많이 발생해서 코드 분석하고 고치는데 하루 정도 걸린 것 같다. 서비스타임이 끝났는데도 만약에 해당 프로세의 티켓 범위의 랜덤수가 뽑히면 그 프로세스가 계속 나와서 정말 어떻게 해결해야 할지 막막했었는데, 오랜 수정 작업과 분석 끝에 저러한 경우가 생기면 counter 와 티켓을 다음 티켓으로 초기화해주는 방식으로 해결을 했다. 이론적으로 되게 단순한 해결방안 이였는데, 역시 코드가 길어지면서 조건문의 약간의 논리적 오류나, 순서가 잘못되면 결과가 잘못 나오는 일이 진짜 생각보다 많이 생겼다.

이번 과제를 하면서 교수님이 왜 팀 프로젝트로 이번 과제를 내주신지 이해가 됐다. 나 혼자 이번 프로젝트를 했으면 둘이 할 때 보다 훨씬 오랜 시간이 필요했을 것 같다. 동학이와 구글 , 네이버 이미 만들어진 자료들 정말 하나도 찾아보지 않고 오로지 수업시간에 배운 원리만으로 거의 20 시간정도를 알고리즘을 짜고 시도해보고 안되면 수정하는 작업을 진짜 수없이 반복한 끝에 결과물을 만들어 냈다는게 정말 뿌듯하다. 아쉬운 점은 코드가 조건문을 정말 많이 썼고, 반복문, 논리 연산도 너무 많이 쓰게 돼서 코드가 점점 길어지다 보니 코드를 딱 봤을 때 내용을 알아볼 수 있게 구현하지 못하게 된 것과, 도착시간이 1 차이가 나는 것은 해결 했지만, 만약 2 이상 차이가 나게 되면 프로그램이 아예 실행 안되는 문제를 끝까지 해결 못한게 너무 아쉽다.

이론적 지식을 현실에 적용해서 실행 시키는 것이 현실과는 다르다는 것을 너무 많이 느끼는 과제였고 과제를 하는 동안 많은 어려움도 있고 진짜 포기하고 싶었던 순간도 많았지만 동학이와 끝까지 문제 해결하고 수정하고 했던 20 시간이 넘는 시간들이 헛되지 않고 너무 뿌듯하게 돌아온 것 같다.