# 운영체제

# HW #2

- Lab 1


과목명 | 운영체제

분반 | 2분반

담당 교수 | 최종무 교수님

학과 | SW융합경제경영

학번 | 32151906

이름 | 박준형


제출일 | 2022.04.08

# 개요

**과제 목표**

가) FIFO, RR(q=1, q=4), SPN, HRRN, MLFQ(q=1, q=2^i) 스케줄링 정책을 시뮬레이터로 구현

나) 보너스: 로터리 스케줄링 구현

**과제 산출물**

가) 스케줄링 시뮬레이터 소스 코드

나) 레포트

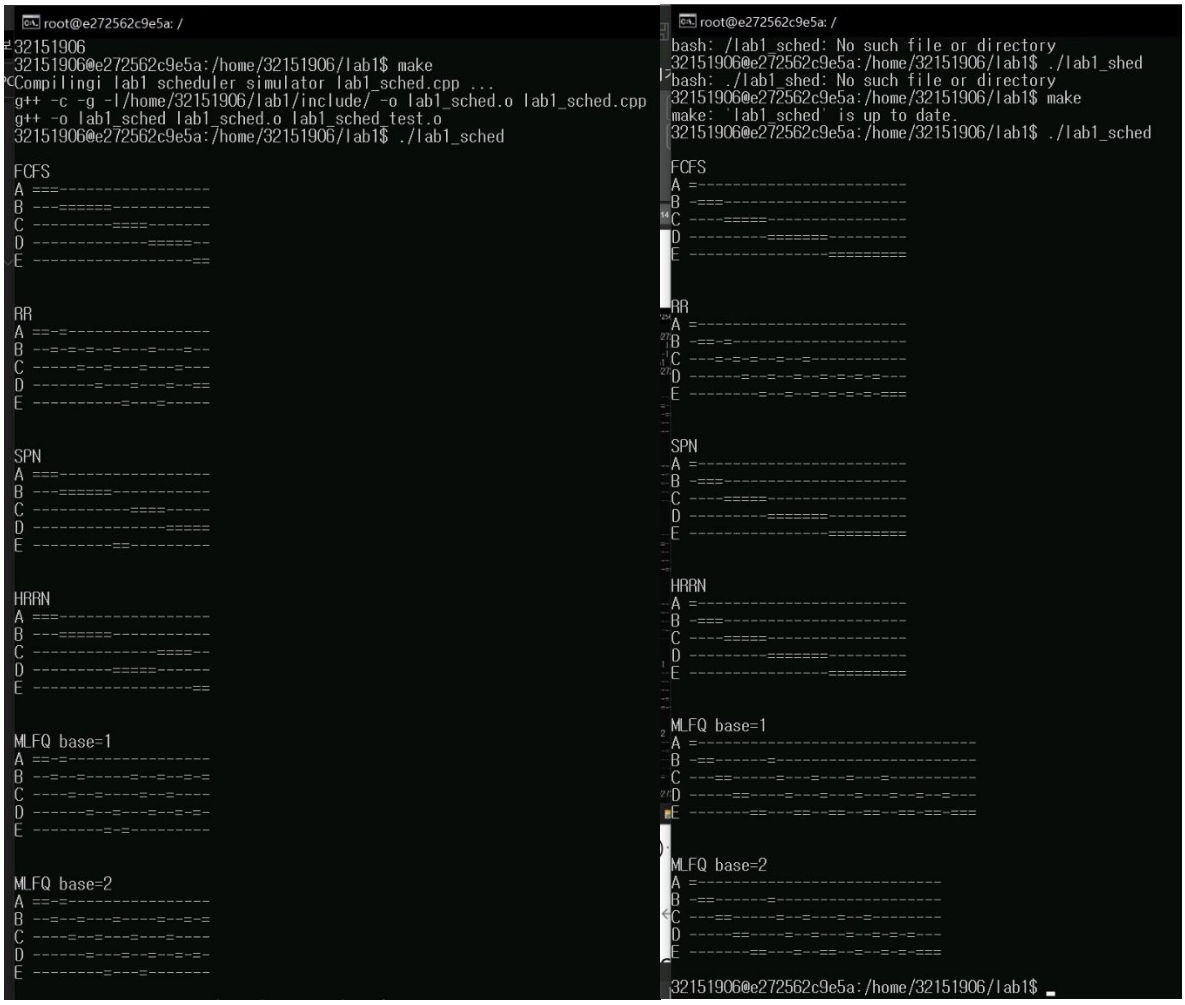
**개요**

본 레포트는 C++로 구현한 스케줄링 시뮬레이터의 실행 결과를 설명한 후 고찰을 제시한다.

# 목차

# 스케줄링 시뮬레이터

## 프로그램 실행 결과




(그림 1) 스케줄링 시뮬레이터 실행 결과

## 프로그램 실행 결과 설명

| 프로세스 이름 | 좌측 | | 우측 | |
|---|---|---|---|---|
| | Arrival Time | Service Time | Arrival Time | Service Time |
| A | 0 | 3 | 1 | 1 |
| B | 2 | 6 | 2 | 3 |
| C | 4 | 4 | 4 | 5 |
| D | 6 | 5 | 6 | 7 |
| E | 8 | 2 | 8 | 9 |

(표 1) 스케줄링 시뮬레이터 입력 프로세스 목록

| 스케줄링 정책 | 좌측 | | 우측 | |
|---|---|---|---|---|
| | 실행 결과 | 비고 | 실행 결과 | 비고 |
| FCFS | 정상 | 24 페이지 워크로드 수행 결과와 일치 | 정상 | X |
| RR | 정상 | | 정상 | X |
| SPN | 정상 | | 정상 | X |
| HRRN | 비정상 | ~와 불일치 | - | 기존 워크로드의 정상 작동 확인이 불가함으로 실행 결과 공란 |
| MLFQ(q=1) | 정상 | ~와 일치 | 비정상 | 제시된 서비스 타임보다 더 많은 시간 동안 작업 수행 |
| MLFQ(q=2^i) | 비정상 | ~와 불일치 | - | 기존 워크로드의 정상 작동 확인이 불가함으로 실행 결과 공란 |

(표 2) 실행 결과 설명

# 고찰

**C++**

C++을 공부해볼 겸 해서 C++로 구현해봤는데, 확실히 C보다 편한 부분이 많았던 것 같다. 비록 지금도 과제를 온전히 다 하지는 못했지만, 만약에 C로 과제를 구현했다면 지금보다도 진행도가 더 떨어지는 결과물을 냈을 것 같다. C++로 구현하는 동안 확실히 편하기는 했지만, 한가지 문제가 바로 구현하면서 새로운 것들을 알아가다보니 코드가 일관성을 잃는다는 점이었다. 가령, 처음에는 프로세스 클래스를 만들어 프로세스 객체에 모든 정보를 저장해두고 이를 활용해서 스케줄링을 하려고 했는데, 튜플이나 sort 등의 존재를 알고 나서는 그냥 튜플로 데이터를 다루는 게 더 편한 부분이 많아서 후반부의 코드는 원래의 의도와는 다르게 구현을 시도하게 되었고, 의도가 바뀌었음에도 초기의 의도가 담긴 맥락의 코드에 자잘한 수정을 가하는 형식으로 코드가 작성되다 보니, 뒷부분의 스케줄링 정책들의 난이도가 더 높았던 것을 감안하더라도 후반부로 갈수록 코드 작성이 힘들어졌던 것 같다.

**메모리 누수**

직접적으로 확인해보지는 못했지만, 아마 현재의 코드에서는 메모리 누수가 발생할 여지가 많을 것으로 보인다. C++과 관련해서 이런저런 검색을 하다보니 포인터를 사용하는 과정에서 포인터가 가르키던 데이터를 메모리 해제없이 변경하면 기존의 데이터가 메모리에 고스란히 남아 메모리 누수로 이어질 수도 있다는 이야기가 있었는데, 이는 파이썬과 같이 메모리를 자동적으로 처리해주는 언어에 익숙하다보니 간과한 부분이었다. 이 메모리 누수과 관련한 내용을 발견한 것은 이미 과제를 상당부분한 상태인데다가 제출이 얼마 남지 않은 시점이었기에, 부득이하게 이를 해결하지 못했다.

**MLFQ의 큐 레벨 변경**

과제를 온전히 다 완성하지 못했기에 가장 어려웠던 부분을 꼽기는 애매하지만, 가장 스트레스 받았던 부분을 꼽자면 단 하나의 프로세스만이 스케줄링되고 있다가 새로운 프로세스가 들어오는 시점에서 이루어지는 기존 프로세스의

큐 레벨 변경이었다. 기존의 모든 스케줄링 정책을 구현할 때, 대체로 한 타임 퀀텀 동안의 실행이 끝난 것으로 간주된 이후에 이런저런 처리를 하는 식으로 진행해왔는데, 해당 상황의 경우 프로세스가 실행된 것으로 간주된 후가 아닌 프로세스가 실행되기 전과 새로운 프로세스가 들어온 이후 그 사이에 큐 레벨 변경을 해줘야 하는 것이라 기존의 맥락을 수정해야하는 것이 까다로웠다. 0에 아무 프로세스도 도착하지 않으면 무한 반복문을 도는 등 여러 시행착오를 겪었는데, 결국 실행 결과가 비정상인 것을 보면 이 부분도 제대로 보완하지는 못한 것 가다.

**클린 코드**

얼마전에 클린 코드라는 책을 읽어보고 난 뒤에, '코드는 가장 상세한 기술 명세서이다' 라는 말과 신문 기사를 읽듯이 읽힐 수 있어야 한다는 말이 참 와닿았어서 그렇게 작성하려고 노력해보았다. 변수 명을 보면 무슨 역할인 지 알고, 함수 명을 보면 무슨 기능인 지 알도록 이름을 상세히 짓고, 논리를 같은 레벨에서 추상화해서 자연스럽게 논리를 따라갈 수 있도록 하고 그런 것들을 해보려고 했는데, 결국 점차 마음이 급해지면서 결국엔 되는 대로 작성하게 된 것 같아서 아쉬운 부분이 많다. 당장 무언가를 비교해서 조건문을 작성할 때도, 직접 변수끼리 비교하는 것이 아니라 클래스의 메서드로 추상화할 수 있었을 것이다.

**그 외**

여러모로 아쉬운 부분이 많았는데, 일단은 최적화된 결과는 아니더라도 최소한의 만족할 만한 기준의 결과조차도 내지 못한게 아쉽고, 또, 로터리 스케쥴링이 가장 구현이 재밌을 것 같다는 생각을 했었는데 이를 구현해보지 못한 점도 아쉬우며, RR에 옵션을 줄 수 있도록 구현해야 한다는 것을 뒤늦게 알아서 1칸의 타임 퀀텀만 가능한 RR밖에 구현하지 못한 것도 아쉽다. 그리고 보다 공을 들여 깔끔한 코드를 작성하지 못한 것도... C++에 조금만 더 익숙했으면 시간 내에 구현도 하고, 아마 더 많은 것을 배울 수 있었을 것 같다는 생각이 든다. 그것과는 별개로 이번에 C++에 대해 알아보면서 C++의 장점에 대해 많이 느껴서, C++로 알고리즘 공부를 해나갈 생각이다.