CPU Scheduling Algorithms

이름 : 권영준

소속 명세 : Student, Dept. of Computer Engineering, Yeungnam University, Gyeongsan, Korea

[요 약]

본 프로젝트에서는 CPU Scheduling 알고리즘이 필요한 이유와 종류를 설명하고 각각의 알고리즘의 특징들을 시뮬레이터를 통해 만들어진 자료를 통해 분석하여 알 수 있도록 한다.

▸ **주제어** : CPU 스케쥴링, FCFS, SJF, SRTF, Priority, Round Robin, Upgraded Round Robin

I. 서론

본 프로젝트의 개요 및 의의

본 프로젝트의 주요 목표

본 프로젝트의 핵심 제공 기능

II. 배경 지식 및 관련 기술

1. 배경 지식

본 프로젝트를 진행하기 위해 필요한 배경 지식

본 프로젝트 문서를 참고하는 사람이
기본적으로 알고 있어야 하는 배경 지식 (소양)

2. 관련 기술

본 프로젝트에서 분석하는 기술에 대한 현 상황

유사 기술 혹은 이전 기술에 대한 소개

III. (본론) 주요 주제명

1. 주요 주제의 개요

본 프로젝트에서 수행하고자 하는 핵심 주제의 개요

주제 설명을 위한 구조도 제시

2. 주요 주제의 핵심 알고리즘 및 기능

주요 주제의 기능 및 동작 명세

Pseudo 코드(혹 순서도)를 활용한 알고리즘 구체화

알고리즘 이해를 위한 수식 제공 (필요시)

3. 주요 주제의 알고리즘 동작 사례

제공한 알고리즘의 구체적이고 다양한 동작 사례 제시

다이어그램등을 활용한 동작 절차 소개

알고리즘 기반 성능 예측 및 성능 추이에 대한
정량적 분석 (수식 분석을 통한 성능 예측)

IV. 성능 평가

1. 실험 환경

본 프로젝트의 실험 환경 명세
실험 도구, 구현 환경, 구현 언어 등

실험 제약 사항 명세
본 실험만의 특징적 고려 사항 혹은 제약 사항 명세

성능 평가를 위한 입력 요소
해당 입력 요소 선정의 적합성 및 합리성 서술

성능 평가를 위한 접근 방법, 즉 출력 요소
성능 평가 기준 제시
해당 기준 선정의 적합성 및 합리성 서술

2. 실험 결과 및 분석

제공된 실험 환경을 통해 도출된 구체적인 실험 결과
엑셀 차트 혹은 표를 활용한 정량적 결과 제시

그래프나 표로 제공된 실험 결과에 대한 분석 서술
결과의 변화 추이에 대한 원인 분석
가변 인자 변경에 따른 결과 변화도 상관 분석
정성적 결과와 정량적 결과의 차이 및 상관도 분석

V. 결론

본 프로젝트의 목표, 기능, 알고리즘, 실험 결과에 대한 간략한 재명세

본 프로젝트에서 도출된 주요 핵심 결론 제시

향후 알고리즘 개선 방안(가능성) 및 추가 실험 요소(필요시) 제시

참고 자료

[1] 본 문서의 주요 참고 자료 제공
[2] 도서, 웹 싸이트, 기술 문서등
문서 내부에 index를 붙여 직접 참조가 가능하도록 편집

I. 서론

운영체제는 프로세스를 관리한다. CPU가 한번에 실행할 수 있는 프로세스는 하나이다. CPU Scheduling은 프로세스가 작업을 수행할 때, 언제 어떤 프로세스가 CPU에 할당될지를 결정하는 작업이다. 수많은 프로세스를 효율적으로 처리하기 위해서는 성능이 좋은 알고리즘이 필요하다. 본 프로젝트는 FCFS, SJF, SRTF, Priority, Round Robin 등 CPU Scheduling 알고리즘을 직접 구현하여 CPU Scheduling을 이해하고 또 새로운 알고리즘을 고안해내는 것을 목표로 한다. 본 프로젝트는 CPU Scheduling 알고리즘을 설명하고 CPU Scheduling 알고리즘의 성능을 보여주어 알고리즘 간의 성능을 보기 쉽게 전달한다.

II. 배경 지식 및 관련 기술

1. 배경 지식

본 프로젝트의 필요한 배경 지식으로는 CPU는 한 번에 하나의 프로세스만 실행할 수 있는 것과 CPU Scheduling 알고리즘의 기준과 선점형과 비선점형의 개념이다.

그 기준으로는 CPU 활용률(CPU utilization), 처리량(Throughput), 응답 시간(Response Time), 대기 시간(Waiting Time), 반환 시간(Turnaround Time)이다. 먼저, CPU 활용도이다. CPU 활용도는 전체 시스템 시간 중 CPU가 작업을 처리하는 시간의 비율이다. CPU 활용도는 높을수록 좋다. 다음으로 처리량은 CPU가 단위 시간 당 처리하는 프로세스의 개수이다. 이 수치 역시 높을수록 좋다. 다음으로는 응답 시간이다. 응답 시간은 프로세스가 입출력을 시작해서 첫 결과가 나오는데 걸리는 시간이다. 앞서 말한 거와 달리 이 수치는 낮을수록 좋다. 다음으로 대기 시간은 프로세스가 Ready Queue에서 대기하는 시간의 총합이다. 이 수치는 응답 시간과 같이 낮을수록 좋다. 마지막으로 반환 시간은 프로세스가 시작해서 끝날때까지 걸리는 시간이다.

비선점형은 새로 Ready Queue에 들어와도 기존에 실행하는 프로세스는 종료될 때까지 실행하는 것이고 선점형은 실행중이라도 다른 프로세스에 의해 Swap out될 수 있는 것이다.

2. 관련 기술

Operating System Concepts에서 설명하는 CPU Scheduling 알고리즘의 종류로는 FCFS, Shortest Job First Scheduling, Shortest Remaining Time First Scheduling, Priority Scheduling, Round Robin, Multilevel Queue Scheduling, Multilevel Feedback Queue Scheduling, Thread Scheduling, Multi-Processor Scheduling, Real Time Scheduling, Deadline based Scheduling이 있다. 대표적인 알고리즘의 관련 설명은 아래 표와 같다.

	스케쥴링 방법	**특징**
FCFS	도착 순서를 기준으로 처리	간단하고 공평함 처리시간 예측가능
SJF	실행시간이 가장 짧은 작업을 먼저 처리	짧은 작업에 유리함 긴 작업은 대기 시간이 길
SRTF	실행 중 더 짧은 작업이 존재하면 먼저 처리	실행시간의 지속적 추적 필요 긴 작업은 SJF보다 대기 시간이 더 길어짐
비선점형 우선순위	우선순위가 높은 순서대로 처리	Starvation 현상 발생
선점형 우선순위	실행 중 우선순위가 더 높은 작업이 존재하면 먼저 처리	Starvation 현상 발생
라운드 로빈	Time Slice를 기준으로 처리	적절한 Time Slice의 탐색과 잦은 문맥교환 오버헤드 발생

표 1 CPU 스케쥴링 알고리즘의 종류

CPU Scheduling은 계속 연구되고 있는 분야로 이외에도 수많은 정책들이 있다.

III. CPU Scheduling Algorithms

1. CPU Scheduling Algorithms의 개요

CPU Scheduling Algorithms은 프로세스를 효율적으로 관리하기 위해 필요하다, CPU는 한번에 하나의 프로세스만 처리할 수 있으므로 수많은 프로세스를 기준 없이 막무가내로 처리하게 되면 성능이 낮아질 수 있다. 그러므로 앞서 기술한 FCFS, SJF. Priority 등 수많은 알고리즘이 개발되었다.

본 프로젝트의 알고리즘의 성능 평가는 응답 시간, 대기 시간, 반환 시간, CPU 사용률을 기준으로 평가한다. 시뮬레이터의 순서도는 다음 그림과 같다.

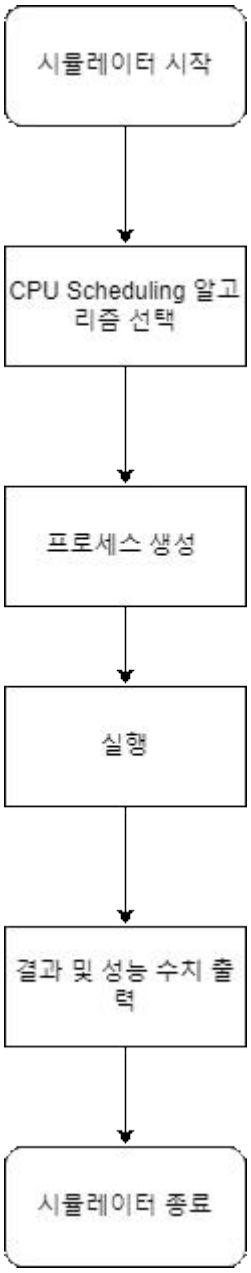


그림 1 시뮬레이터의 전체 순서도

2. CPU Scheduling Algorithms 핵심 알고리즘 및 기능

CPU Scheduling 알고리즘의 종류로는 FCFS, Shortest Job First Scheduling, Shortest Remaining Time First Scheduling, Priority Scheduling, Round Robin 등이 있다. FCFS는 도착 순서를 기준으로 먼저 도착한 프로세스를 실행한다. SJF는 레디큐에 있는 프로세스 중 실행시간이 가장 짧

은 프로세스를 먼저 처리한다. SJF의 단점은 실행시간이 긴 프로세스는 대기 시간이 길어지게 된다. SRTF는 선점형 SJF로서 실행 중인 프로세스보다 더 짧은 작업이 존재하면 먼저 처리한다. 실행시간을 지속적으로 파악하여야 하고 역시 실행시간이 긴 프로세스는 대기 시간이 길어지게 된다. Priority Scheduling은 프로세스의 우선순위를 기준으로 실행한다. 비선점형 Priority Scheduling은 실행 중에 레디큐에 우선순위가 높은 프로세스가 도착하여도 지금 실행 중인 프로세스를 계속 실행한다. 선점형 Priority Scheduling은 실행 중인 프로세스보다 우선순위가 높은 프로세스가 도착하면 실행 중인 프로세스를 중단하고 우선순위가 높은 프로세스를 실행한다. Priority Scheduling의 단점은 Starvation 현상이다. 우선순위가 낮은 프로세스는 실행을 못 할 수도 있게 된다. 이를 해결하기 위한 기법으로는 시간이 흐를수록 우선순위를 증가시키는 개념인 aging이 있다. Round Robin은 특정 time slice만큼 프로세스에게 할당하는 알고리즘이다. 반환 시간을 최적화하는 time slice는 없어 적절한 time slice를 찾는 연구는 계속되고 있다. 또한 잦은 문맥교환으로 인해 오버헤드가 발생한다.

각 알고리즘의 Pseudo 코드는 다음과 같다.

//FCFS의 Pseudo 코드

1. 프로세스들을 입력받는다.
2. 프로세스의 개수를 구한다.
3. 도착시간 순으로 정렬한다.
4. 현재시간이 도착시간과 같으면 실행한다.
5. 실행이 끝나면 완료된 프로세스의 개수를 증가시킨다.
6. 실행 중 도착한 프로세스가 있는지 찾는다.
7. 있으면 실행하고 5번으로 간다.
8. 없으면 현재시간을 증가한다.
9. 완료된 프로세스의 개수가 프로세스의 개수와 같으면 끝낸다.

//SJF의 Pseudo 코드

1. 프로세스들을 입력받는다.
2. 프로세스의 개수를 구한다.
3. 도착시간이 현재시간과 같은 프로세스를 찾는다.
4. 여러 개 있으면 실행시간을 비교하여 실행시간이 짧은 프로세스를 찾는다.
5. 찾은 프로세스를 실행한다.
6. 현재시간을 실행한 프로세스의 실행시간만큼 증가시킨다.
7. 실행이 끝나면 완료된 프로세스의 개수를 증가시키고 도착한 프로세스가 있는지 찾는다.
8. 있으면 그중에서 실행시간이 짧은 프로세스를 실행한다.
9. 없으면 현재시간을 증가한다.
10. 완료된 프로세스의 개수가 프로세스의 개수와 같으면 끝낸다.

//SRTF의 Pseudo 코드

1. 프로세스들을 입력받는다.
2. 프로세스의 개수를 구한다.
3. 도착시간이 현재시간과 같은 프로세스를 찾는다.
4. 여러 개 있으면 실행시간을 비교하여 실행시간이 짧은 프로세스를 실행한다.
5. 실행 중 현재 실행 중인 프로세스의 남은 시간보다 더 짧은 프로세스가 있는지 찾는다.

6. 있으면 실행 중인 프로세스를 멈추고 더 짧은 프로세스를 실행한다.
7. 없으면 계속 실행한다.
8. 실행이 끝난 프로세스가 있으면 완료된 프로세스의 개수를 증가시킨다.
9. 완료된 프로세스의 개수가 프로세스의 개수와 같으면 끝낸다.

//Priority Scheduling(Non-preemtive)의 Pseudo 코드
1. 프로세스들을 입력받는다.
2. 프로세스의 개수를 구한다.
3. 도착시간이 현재시간과 같은 프로세스를 찾는다.
4. 여러개 있으면 우선순위를 비교하여 우선순위가 높은 프로세스를 실행한다.
5. 현재시간을 실행한 프로세스의 실행시간만큼 증가시킨다.
6. 실행이 끝나면 완료된 프로세스의 개수를 증가시키고 도착한 프로세스가 있는지 찾는다.
7. 있으면 4번으로 간다.
8. 없으면 현재시간을 증가한다.
9. 완료된 프로세스의 개수가 프로세스의 개수와 같으면 끝낸다.

//Priority Scheduling(Preemtive)의 Pseudo 코드
1. 프로세스들을 입력받는다.
2. 프로세스의 개수를 구한다.
3. 도착시간이 현재시간과 같은 프로세스르 찾는다.
4. 여러개 있으면 우선순위를 비교하여 우선순위가 높은 프로세스를 실행한다.
5. 실행 중 현재 실행 중인 프로세스의 우선순위보다 더 높은 우선순위가 있는지 찾는다.
6. 있으면 실행 중인 프로세스를 멈추고 우선순위가 더 높은 프로세스를 실행한다.
7. 없으면 계속 실행한다.
8. 실행이 끝난 프로세스가 있으면 완료된 프로세스의 개수를 증가시킨다.
9. 완료된 프로세스의 개수가 프로세스의 개수와 같으면 끝낸다.

//Priority Scheduling(Preemtive)의 Pseudo 코드
1. 프로세스들을 입력받는다.
2. 프로세스의 개수를 구한다.
3. time slice를 입력받는다.
4. 도착시간이 현재시간과 같은 프로세스를 찾는다.
5. 있으면 해당 프로세스를 time slice만큼 실행한다.
6. 없으면 현재시간을 증가한다.
7. time slice만큼 지나면 실행 중인 프로세스를 멈추고 도착한 프로세스가 있는지 찾는다.
8. 있으면 해당 프로세스를 time slice만큼 실행한다.
9. 없으면 실행 중인 프로세스를 다시 실행한다.
10. 실행이 끝난 프로세스가 있으면 완료된 프로세스의 개수를 증가시킨다.
11. 완료된 프로세스의 개수가 프로세스의 개수와 같으면 끝낸다.

본 프로젝트의 목적 중 새로운 알고리즘을 구현하는 것이 있다. 다음은 새로운 알고리즘의 Pseudo 코드이다. 이름은 Upgraded Round Robin으로 칭한다.

1. 프로세스들을 입력받는다.
2. 프로세스의 개수를 구한다.
3. time slice를 입력받는다.
4. 도착시간이 현재시간과 같은 프로세스를 찾는다.
5. 있으면 해당 프로세스를 time slice만큼 실행한다.
6. 없으면 현재시간을 증가한다.
7. time slice만큼 지나면 실행 중인 프로세스의 남은 시간과 time slice를 비교한다.
8. 작으면 남은 시간까지 실행한다.
9. 크면 실행 중인 프로세스를 멈추고 도착한 프로세스가 있는지 찾는다.
10. 있으면 해당 프로세스를 time slice만큼 실행한다.
11. 없으면 실행 중인 프로세스를 다시 실행한다.
12. 실행이 끝난 프로세스가 있으면 완료된 프로세스의 개수를 증가시킨다.
13. 완료된 프로세스의 개수가 프로세스의 개수와 같으면 끝낸다.

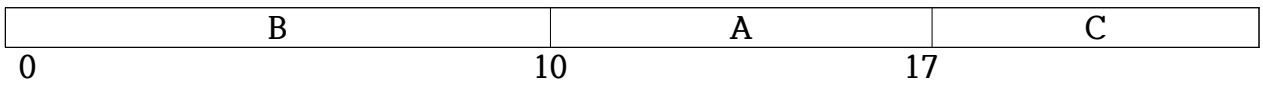
3. CPU Scheduling 알고리즘 동작 사례

프로세스의 상태가 다음 표와 같을 때 알고리즘 별 동작 결과는 간트차트를 통해 표현하면 다음과 같다.

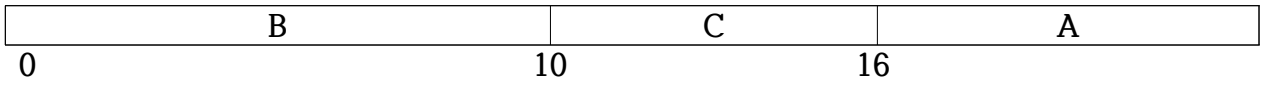
PID	도착시간	실행시간	우선순위
A	1	7	1
B	0	10	3
C	4	6	2

표 2 프로세스의 상태

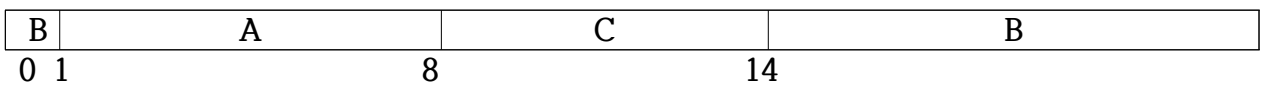
FCFS



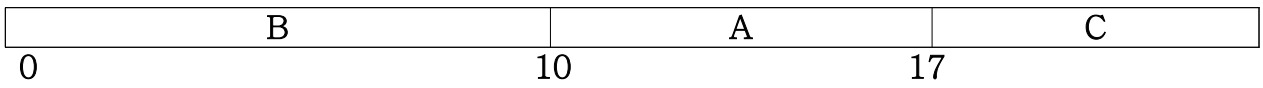
SJF



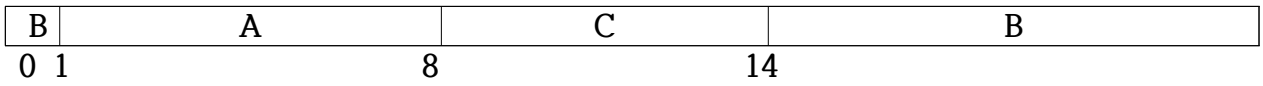
SRTF



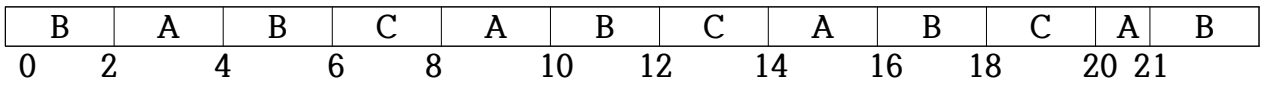
Priority Scheduling(Non-preemtive)



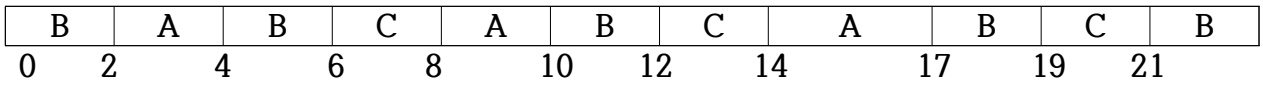
Priority Scheduling(Preemtive)



Round Robin(time slice = 2)



New Algorithms(time slice = 2)



IV. 성능 평가

1. 실험 환경

이 절에서는 시뮬레이터의 구현환경을 설명한다. 개발 언어는 JAVA를 선정하였고 개발 환경은 Eclipse와 JDK 16에서 개발하였다. 성능 평가를 위한 입력 파일은 txt파일이며 txt파일의 양식은 제공한 txt 파일의 양식이다. 자세한 양식은 다음 그림과 같다.

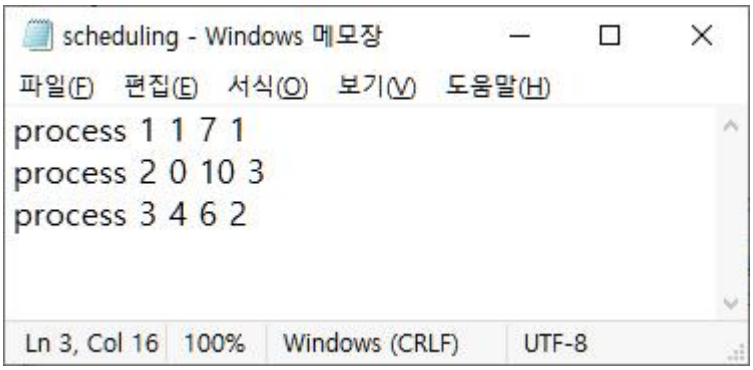


그림 2 입력 파일의 양식

첫 번째 문자열은 process로 적혀있고 2번째 숫자는 PID이며 정수의 오름차순으로 설정하였다. 3번째 숫자는 도착시간(Arrive time), 4번째 숫자는 실행시간(Burst time), 5번째 숫자는 우선순위(Priority)이다. 도착시간은 0 이상의 정수이며 실행시간과 우선순위는 1 이상의 정수이다.

성능 평가는 응답 시간, 반환 시간, 대기 시간, CPU 사용률을 기준으로 평가할 것이고 응답 시간, 반환 시간, 대기 시간은 낮을수록 좋은 것이고 CPU 사용률은 높을수록 좋은 것이다. 응답 시간이 낮을수록 Interactive System에서 빨리 반응하여 사용자의 편의성을 높일 수 있다. 반환 시간은 낮을수록 프로세스가 빨리 종료되었다는 것이므로 낮을수록 좋다. 대기 시간도 마찬가지로 낮을수록 프로세스의 불필요한 시간을 줄이므로 낮을수록 좋다. 마지막으로 CPU 사용률은 높을수록 CPU를 최대한 바쁘게 일하기 때문이다.

실험용 데이터는 엑셀을 이용하여 0을 제외한 랜덤 값으로 이루어진 txt 파일이며 자세한 양식은 다음 그림과 같다.

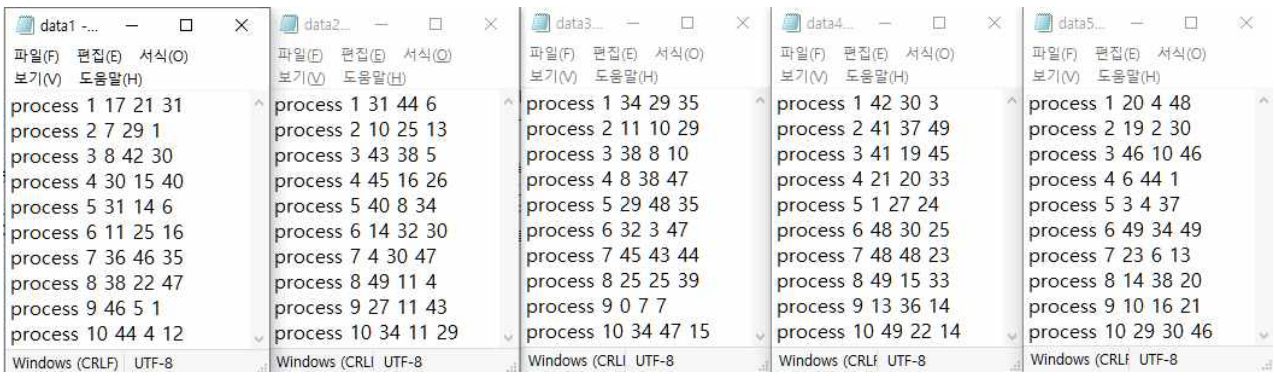


그림 3 실험용 데이터

2. 실험 결과 및 분석

위의 실험용 데이터를 기반으로 FCFS, SJF, SRTF, Priority(Non-preemtive) Scheduling, Priority(Preemtive) Scheduling , Round Robin, Upgraded Round Robin을 시뮬레이터를 통해 실행하였다.

data1의 전체시간은 230, data2의 전체시간은 230, data3의 전체시간은 259, data4의 전체시간은 285, data5의 전체시간은 191이다.

실험용 데이터의 간트 차트는 다음 그림과 같다. 7개의 알고리즘의 5개의 데이터의 간트 차트는 35개이므로 간트차트

는 data1에 대해서만 출력하겠다.

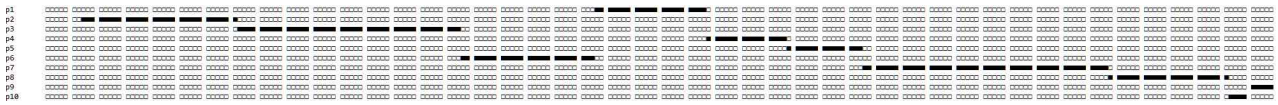


그림 4 data1의 FCFS

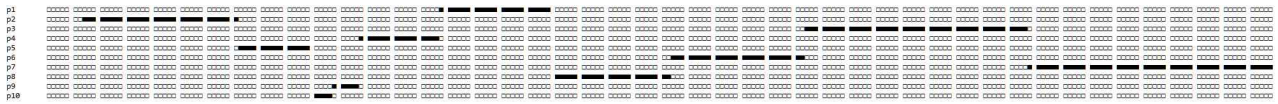


그림 5 data1의 SJF

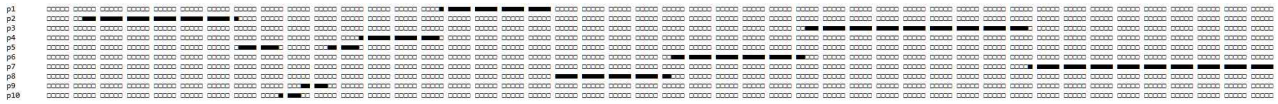


그림 6 data1의 SRTF

그림 7 data1의 Priority(Non-preemtive)

그림 8 data1의 Priority(Preemtive)

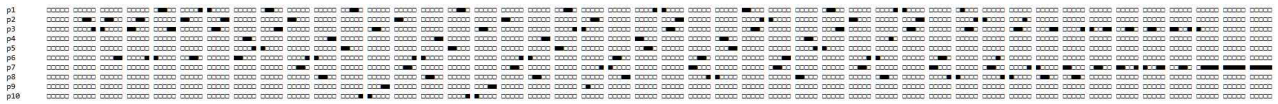


그림 9 data1의 Round Robin

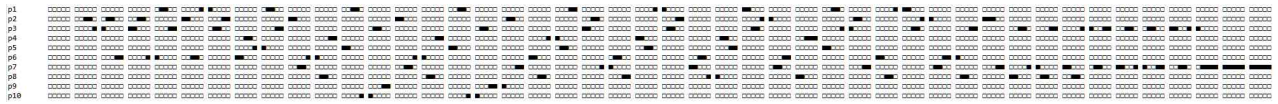
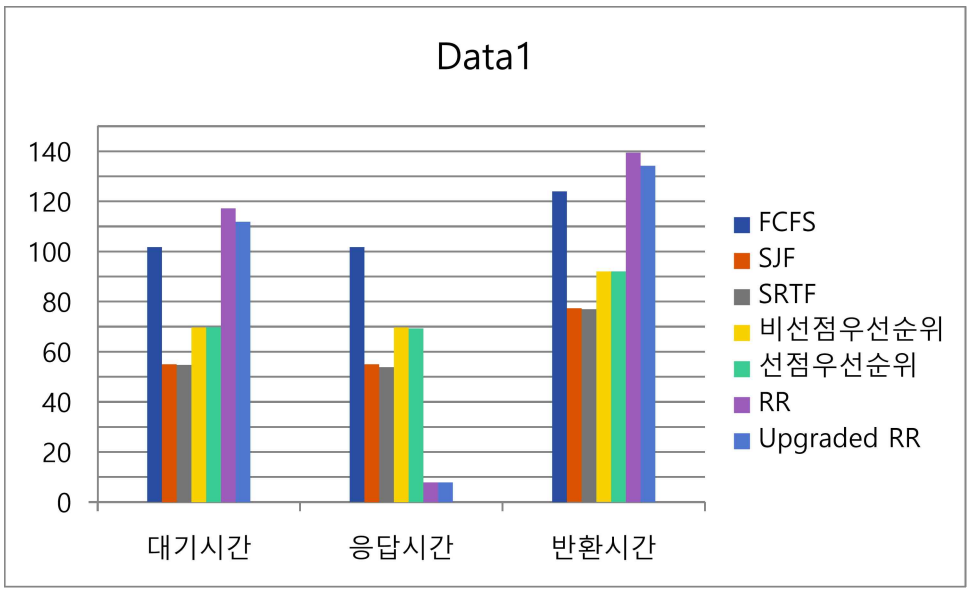
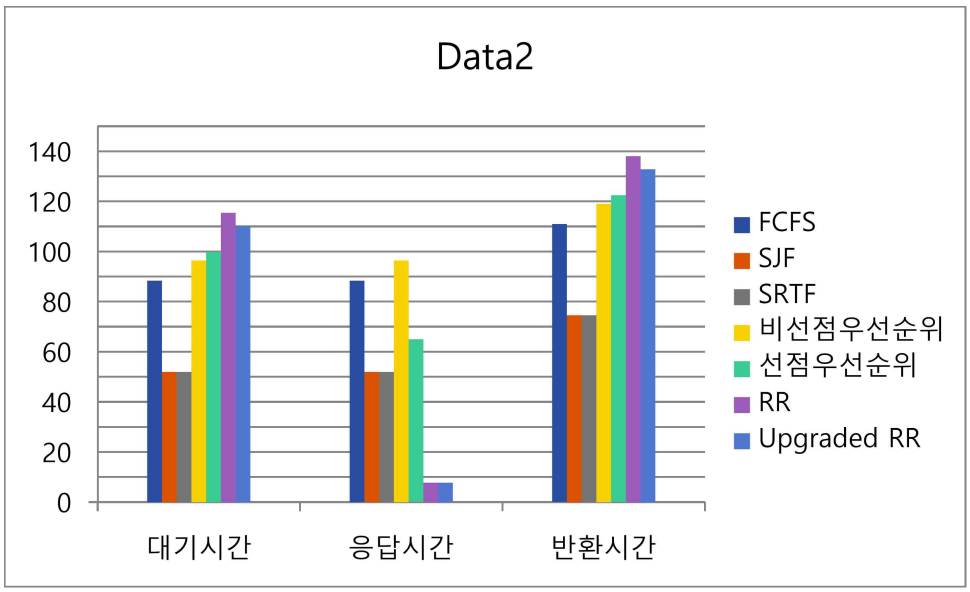


그림 10 data1의 Upgraded Round Robin

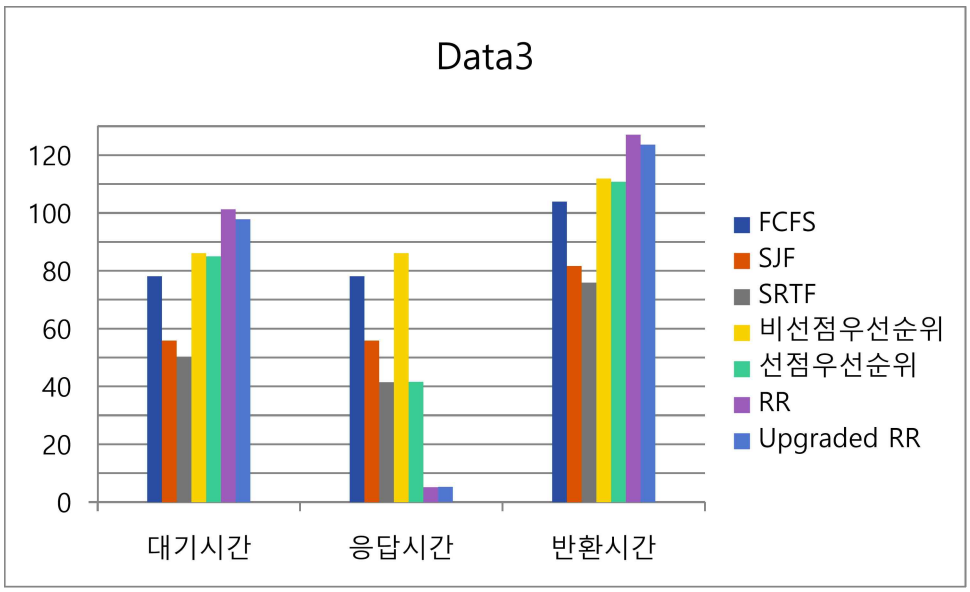
실험용 데이터의 결과를 평가 지표를 기준으로 그린 그래프는 다음과 같다.



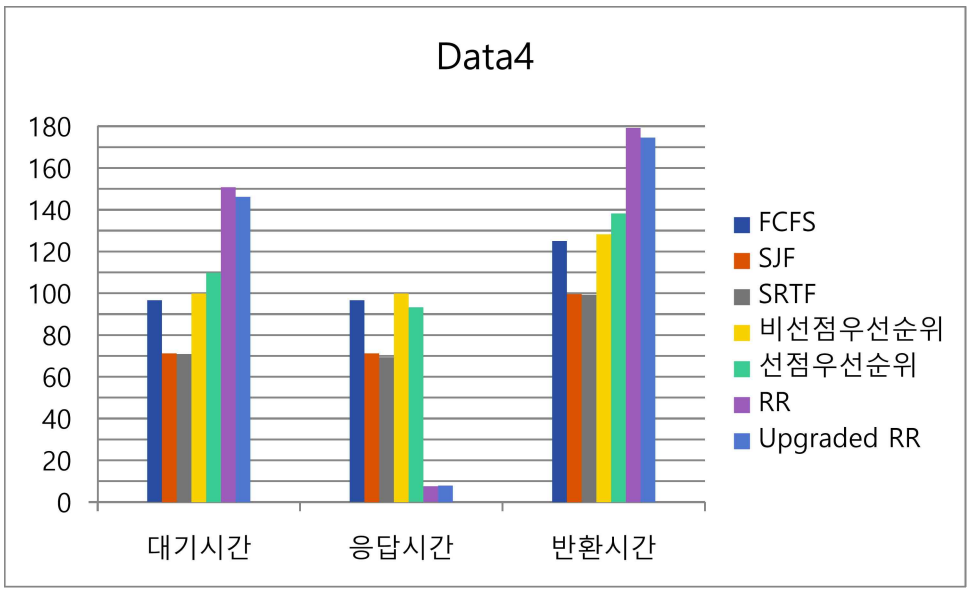
그래프 1 data1의 결과를 표현한 그래프



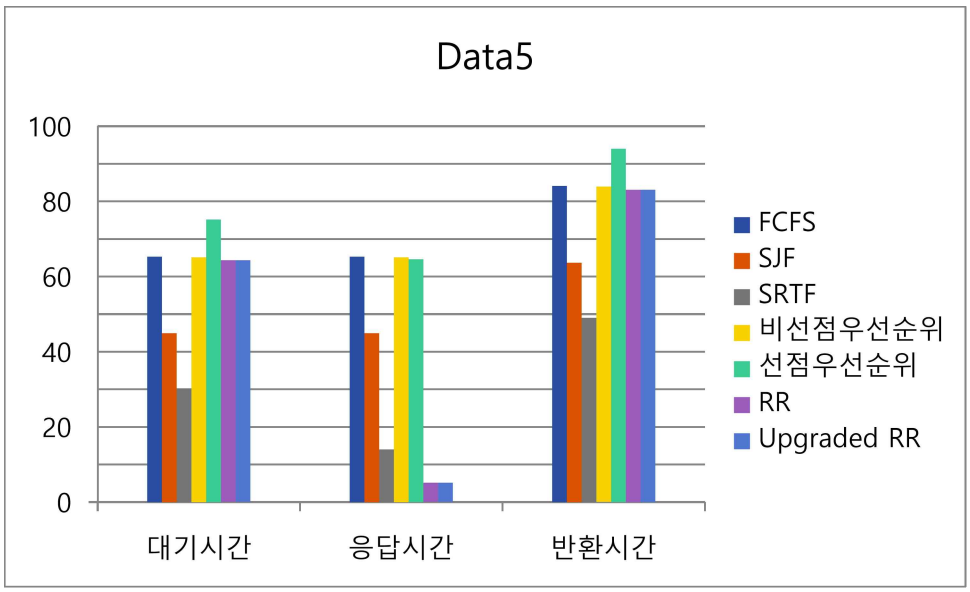
그래프 2 data2의 결과를 표현한 그래프



그래프 3 data3의 결과를 표현한 그래프



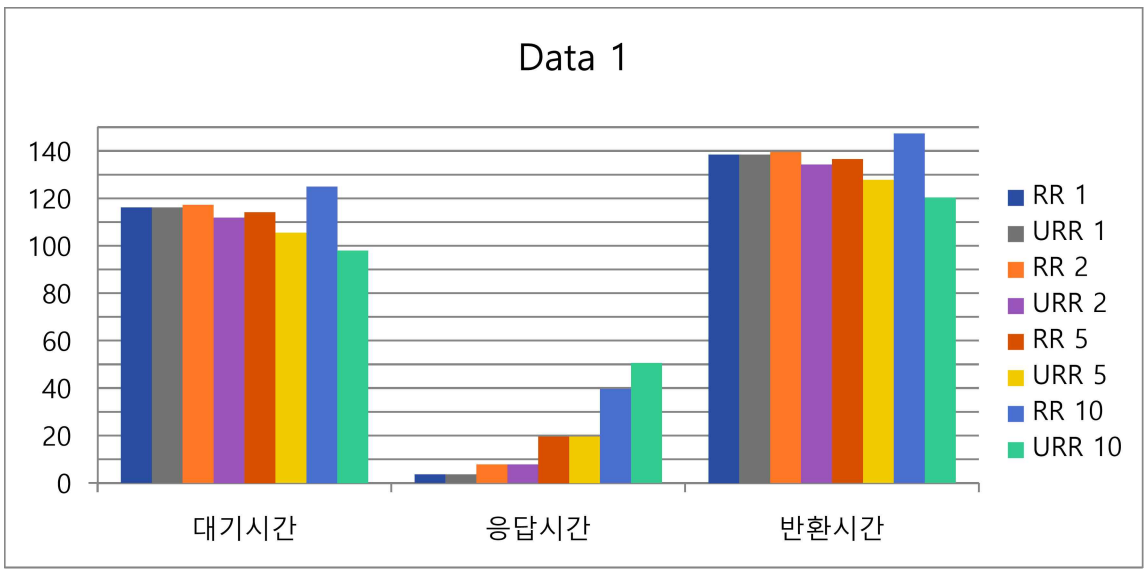
그래프 4 data4의 결과를 표현한 그래프



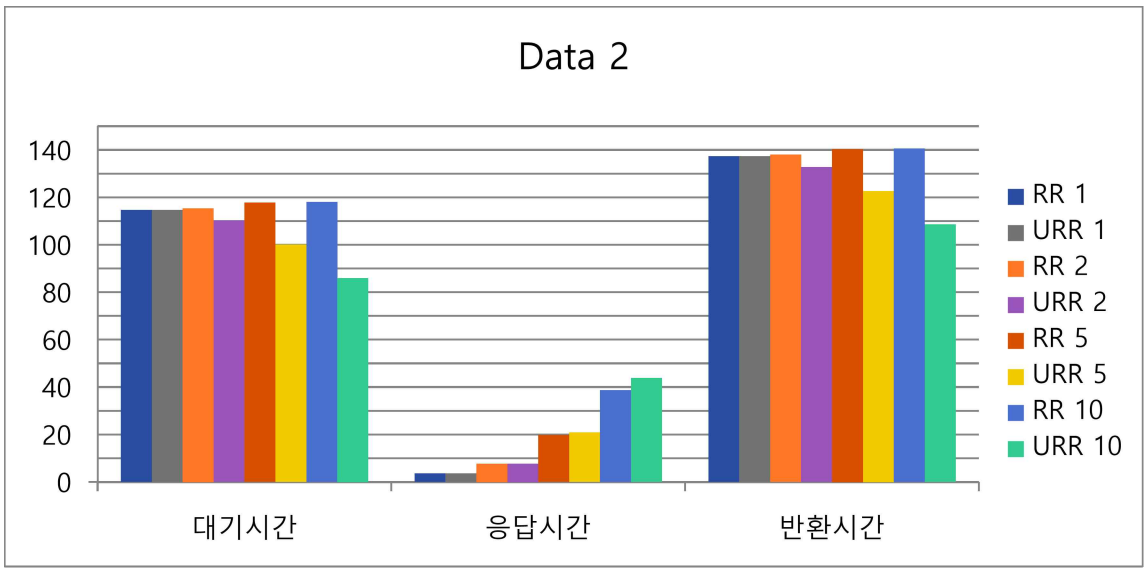
그래프 5 data5의 결과를 표현한 그래프

위의 그래프를 보고 알 수 있는 사실은 Round Robin 계열은 응답 시간은 짧은 반면 대기 시간과 반환 시간은 응답 시간에 비해 훨씬 크다. SJF와 SRTF는 실행시간이 짧은 프로세스를 먼저 실행하므로 다른 알고리즘에 비해 대기 시간과 응답 시간이 짧은 것을 알 수 있다. 반환 시간은 프로세스가 종료될 때까지의 시간이므로 대기 시간과 응답 시간보다 크다는 것을 알 수 있다. 또 반환 시간의 그래프는 대기 시간의 그래프와 매우 유사한 모습을 보이고 있다. 이는 반환 시간은 대기 시간의 값에 프로세스의 실행시간을 더한 것으로 추론할 수 있다.

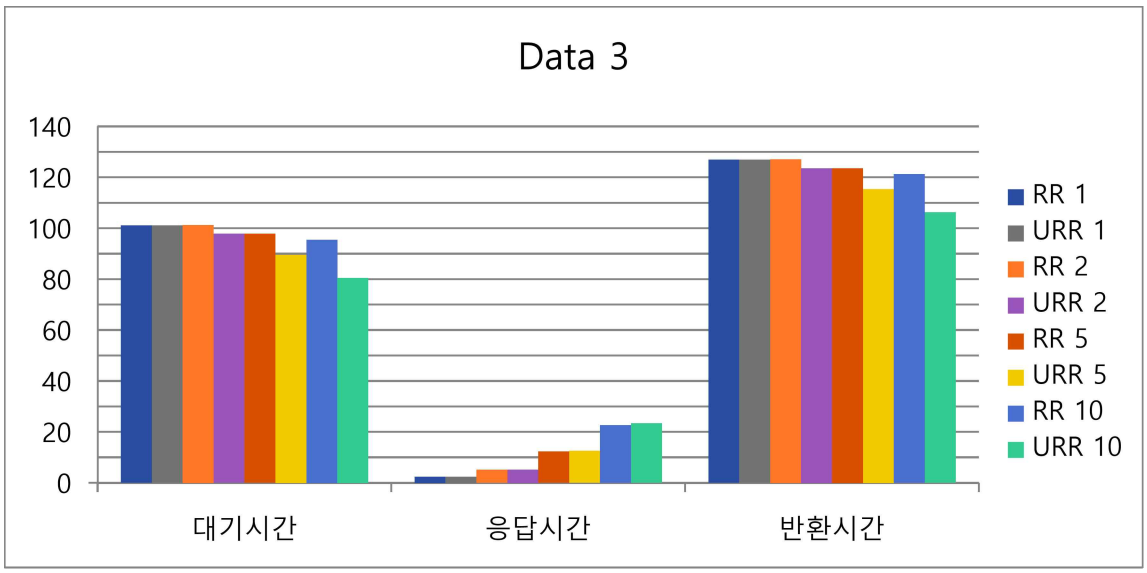
다음은 기존의 Round Robin을 향상시킨 Upgraded Round Robin을 실험용 데이터의 결과를 통해 분석해보았다. 다음 그래프는 Round Robin과 Upgraded Round Robin의 성능지표를 비교한 그래프이다. 숫자는 time slice의 수치이다.



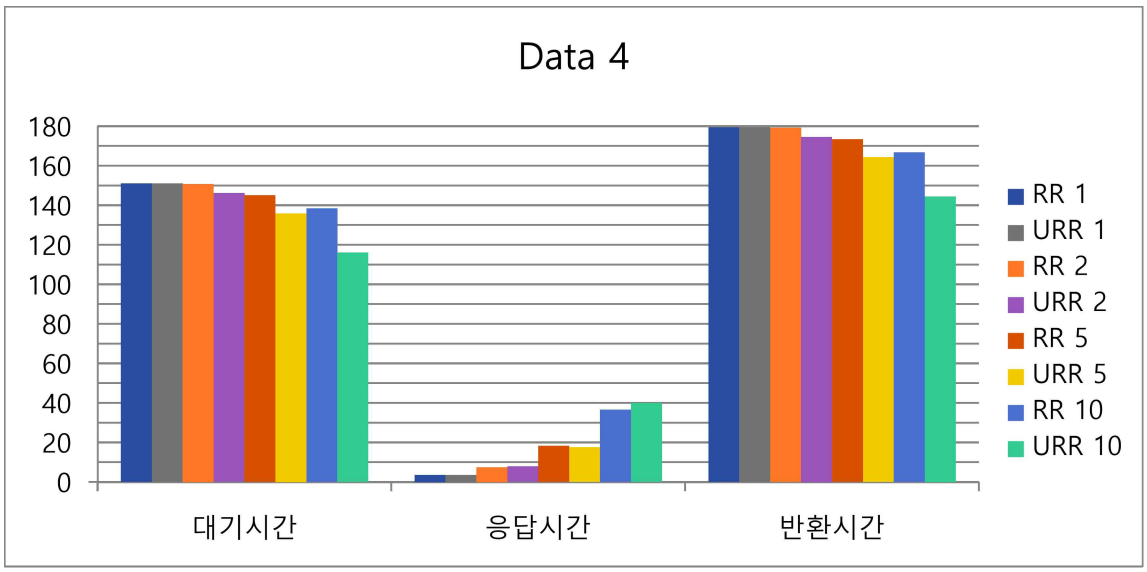
그래프 6 data1을 통한 두 알고리즘의 차이



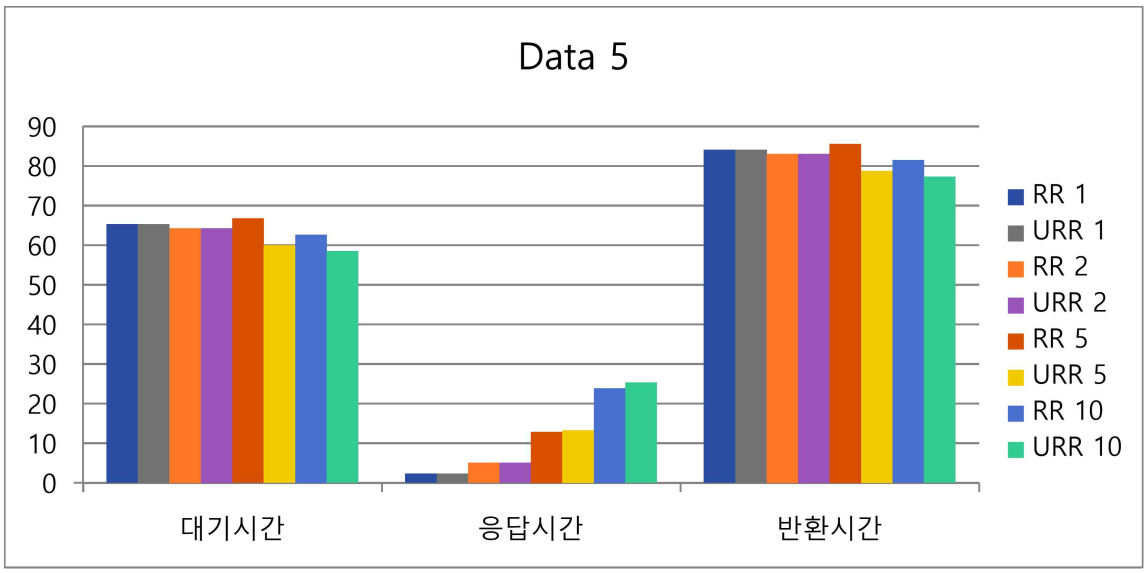
그래프 7 data2를 통한 두 알고리즘의 차이



그래프 8 data3을 통한 두 알고리즘의 차이



그래프 9 data4를 통한 두 알고리즘의 차이



그래프 10 data5를 통한 두 알고리즘의 차이

위의 그래프를 통해 알 수 있는 사실은 time slice가 늘어날수록 응답 시간은 늘어나지만 대기 시간과 반환 시간은 time slice와 관련이 없다는 것이다. 또 Upgraded Round Robin은 기존의 Round Robin에 비해 응답 시간은 미세하게 높은 반면 대기 시간과 반환 시간은 감소하였다는 사실을 알 수 있다.

V. 결론

본 프로젝트는 CPU Scheduling 알고리즘을 설명하여 이해도 증진을 위해 Scheduling 과정을 실시간으로 보여주는 시뮬레이터를 설계하였다. 실험용 데이터를 통하여 시뮬레이션을 실행하여 CPU Scheduling 알고리즘 간의 특징과 간트차트를 통해 알고리즘의 이해도를 높였다. 이를 통해 도출된 결과는 대기시간이 짧은 알고리즘으로는 SJF와 SRTF가 있고 응답시간이 짧은 알고리즘으로는 Round Robin과 Upgraded Round Robin이 있다. 또 반환시간의 그래프는 대기시간의 그래프와 매우 유사하다.

본 프로젝트를 위해 구현한 개발 환경은 Console 기반이지만 향후 GUI로 개발하여 가독성과 시인성을 더하여 이해도를 높이도록 할 것이다. 또한 문맥 교환 시간의 요소도 추가하여 더 현실적인 CPU Scheduling 알고리즘 시뮬레이터로 발전하도록 할 것이다. 향후 Multilevel Queue를 이용한 스케쥴링도 개발하며 CPU Scheduling 알고리즘의 연구를 참조하여 성능이 더 좋은 알고리즘을 개발하도록 노력하겠다. 이를 통해 지속적으로 기능을 개선하면 활용 효과가 극대화될 것이다.

VI. 참고문헌

Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Operating System Concepts 10th Edition