**운영체제 팀프로젝트 최종보고서**

-FCFS를 이용한 스케줄링-



팀 명: THE FUSE

(운영체제 3조)

학 번: 2016150019

성 명: 박재홍 (FCFS)

**1. 스케줄링 알고리즘 선정 이유**

본 설계과제물을 수행하면서 이용한 스케줄링 알고리즘은 First-Come-First-Served 즉, FCFS이다. 그 이유를 꼽자면 First-Come-First-Served 즉, 선착순이야 말로 현실의 사람들이 사는 세계에서도 자주 볼 수 있는 방식이라고 생각했기 때문이다. 수업 시간에도 다뤘듯 이 방식이 비효율적일 수는 있으나, 실상활에서도 자주 이용되는 공정한 방법이기에, 한 번 구현해보고 싶은 마음이 들었다.

**2. FCFS (작성자: 박재홍)**

First-Come-First-Served(이하 FCFS)는 그 이름처럼 선착순으로 처리하는 알고리즘이다. 즉, 먼저 프로세서를 요청한 프로세스에게 먼저 CPU를 할당한다. 이것은 비선점(Non-Preemptive) 방식으로 CPU를 할당 받은 프로세스가 CPU를 반납할 때까지 다른 프로세스들은 기다려야 한다. 다음은 최종적으로 구현한 프로그램의 설명이다. 기본 형식은 e-class에 첨부된 pdf파일을 따랐다. 그렇게 하더라도 프로그램에 대해 설명하는 데에는 아무 문제가 없을 것이라고 판단하였기 때문이다.

**3. 프로그램 구현**

(1) 프로그램의 입력 데이터의 구성

- 프로그램의 입력 데이터는 type, process\_id, priority, computing\_time의 4개 필드로 구성된다.

- type 0은 프로세스의 생성을 나타내며, process-id, priority, computing\_time을 입력 받게 된다.

- type 1은 하나의 프로세스의 시간할당량이 다되어 스케줄이 되어야 함을 나타내며, process\_id, priority, computing\_time의 값은 0이 되며, 특별한 의미는 없다. (시간이 시간할당량 만큼 지난 것을 의미, RR(Round Robin)과 관련된 알고리즘에서 주로 사용)

- type -1은 입력 완료를 의미한다. 입력 완료 이후에는 Queue에 있는 일반 프로세스들이 우선순위에 따라 차례대로 스케줄링 된다.

(2) 생성되는 프로세스의 종류

- 생성되는 프로세스는 일반 프로세스이다.

- 먼저 생성된 프로세스의 우선순위가 나중에 생성된 프로세스의 우선순위보다 높다. (따라서, 파일에서 입력 받는 priority 필드의 값은 여기서 큰 의미가 없다.)

(3) 스케줄링 원칙

- 프로세스는 하나의 Queue로 구성된다.

- 프로세스가 생성되면, 그 순서대로 Queue에 연결된다.

- 프로세스는 Queue에 연결된 순서대로 수행된다. 프로세스의 수행이 끝나면 해당 프로세스의 process\_id, priority, computing\_time, turn\_around\_time, end\_time을 출력한다.

- 스케줄링 및 프로세스 스위칭 오버헤드는 0으로 가정한다.

(4) 프로그램의 출력

- 프로그램은 process\_id, priority, computing\_time, turn\_around\_time, end\_time의 필드로 나뉜 결과를 출력한다.

- process\_id는 파일에서 입력 받은 프로세스의 id를 나타낸다.

- priority는 파일에서 입력 받은 프로세스의 우선순위를 나타낸다. FCFS 방식에서는 큰 의미가 없다.

- computing\_time은 파일에서 입력 받은 값으로, 프로세스의 수행 시간(혹은 시간할당량)을 나타낸다.

- turn\_around\_time은 프로세스의 turn around time 즉, (프로세스 종료 시점) – (프로세스 생성 시점)을 나타낸다.

- end\_time은 프로세스가 종료된 시점을 나타낸다.

(5) 함수, 변수 및 자료 구조(구현 방법)

이중 연결 리스트(Double Linked List)를 이용한 단일 Queue의 형태로 구현하였다.

① 구조체

1. head\_pointer

- Queue의 head와 tail을 저장하는 queue\_pointer 구조체 변수 2개로 구성되어 있다.

b) queue\_pointer

- 생성되는 프로세스의 정보를 입력 받아 저장하는 역할을 한다.

- process\_id, priority, computing\_time, turn\_around\_time, process\_input\_time 변수를 가지고, 다른 프로세스를 가리키는 구조체 변수 2개를 추가로 가지고 있다.

- turn\_around\_time은 지금까지 수행된 computing\_time을 저장한다. (누산)

- input\_file에서 type 1을 입력받을 때 time\_quantum(=20)만큼 증가한다.

② 전역 변수

- input\_time: 각 프로세스가 생성되는 시점을 계산하기 위한 전역 변수

- end\_time: 각 프로세스가 종료되는 시점을 계산하기 위한 전역 변수

③ Queue 초기화(initialize\_queue)

- 프로세스가 들어갈 Queue를 초기화한다.

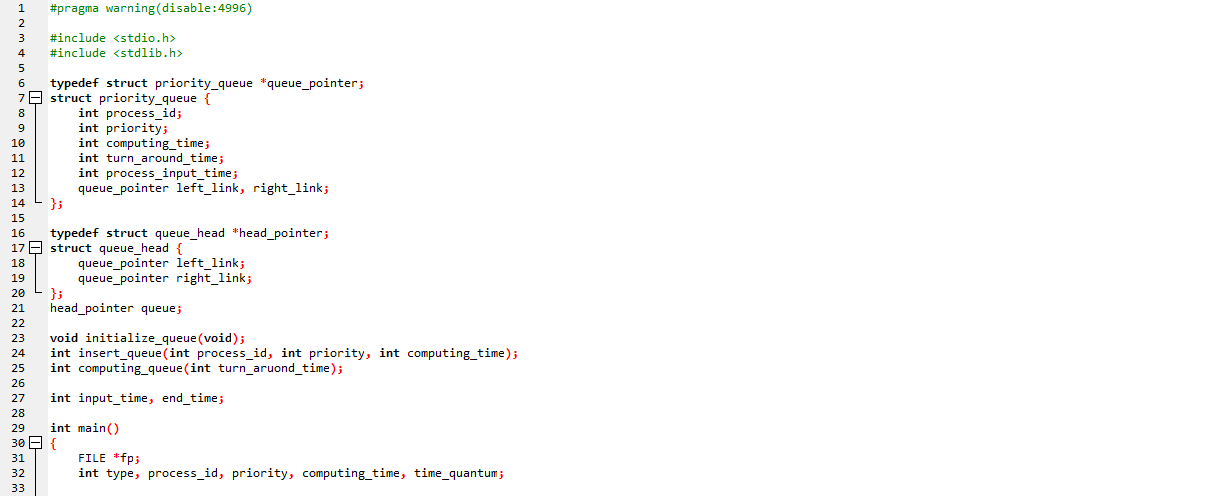
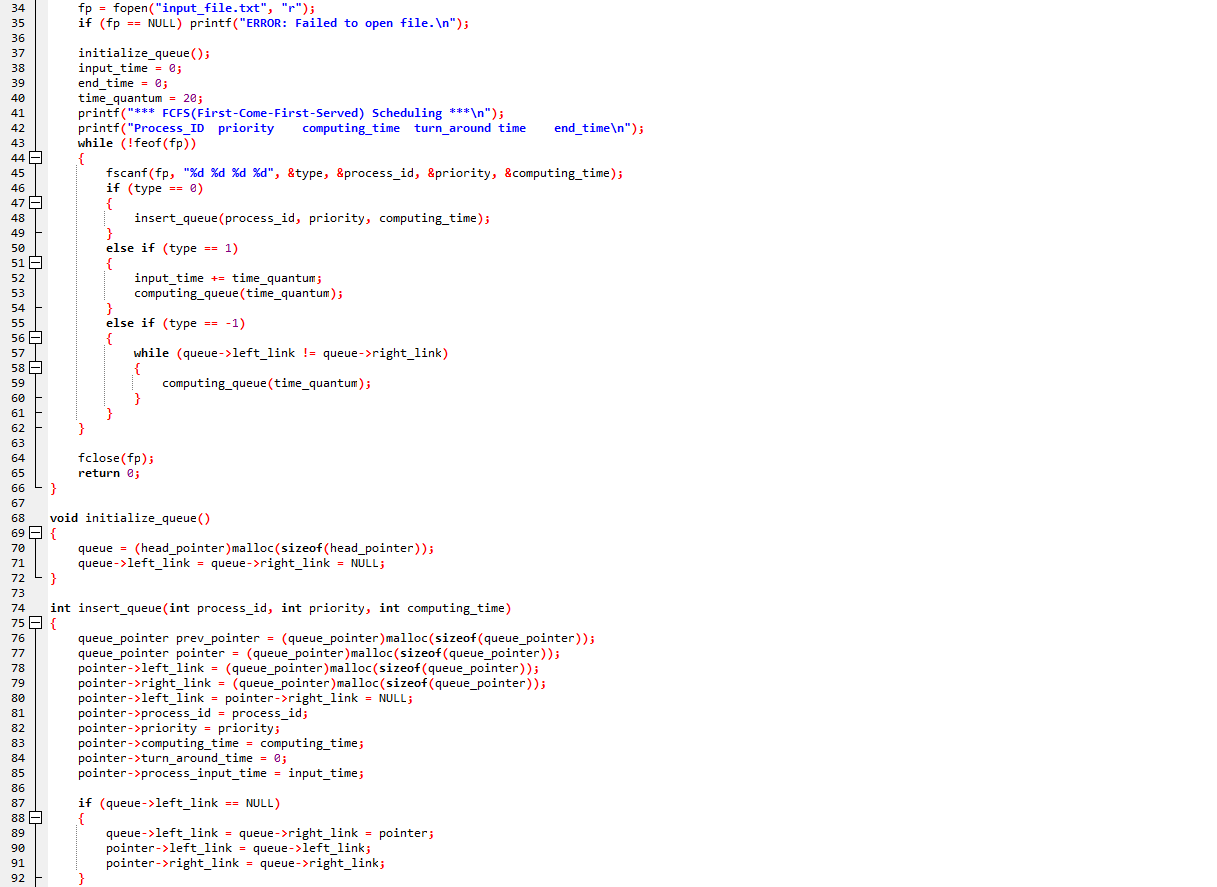
④ 프로세스 생성(insert\_queue)

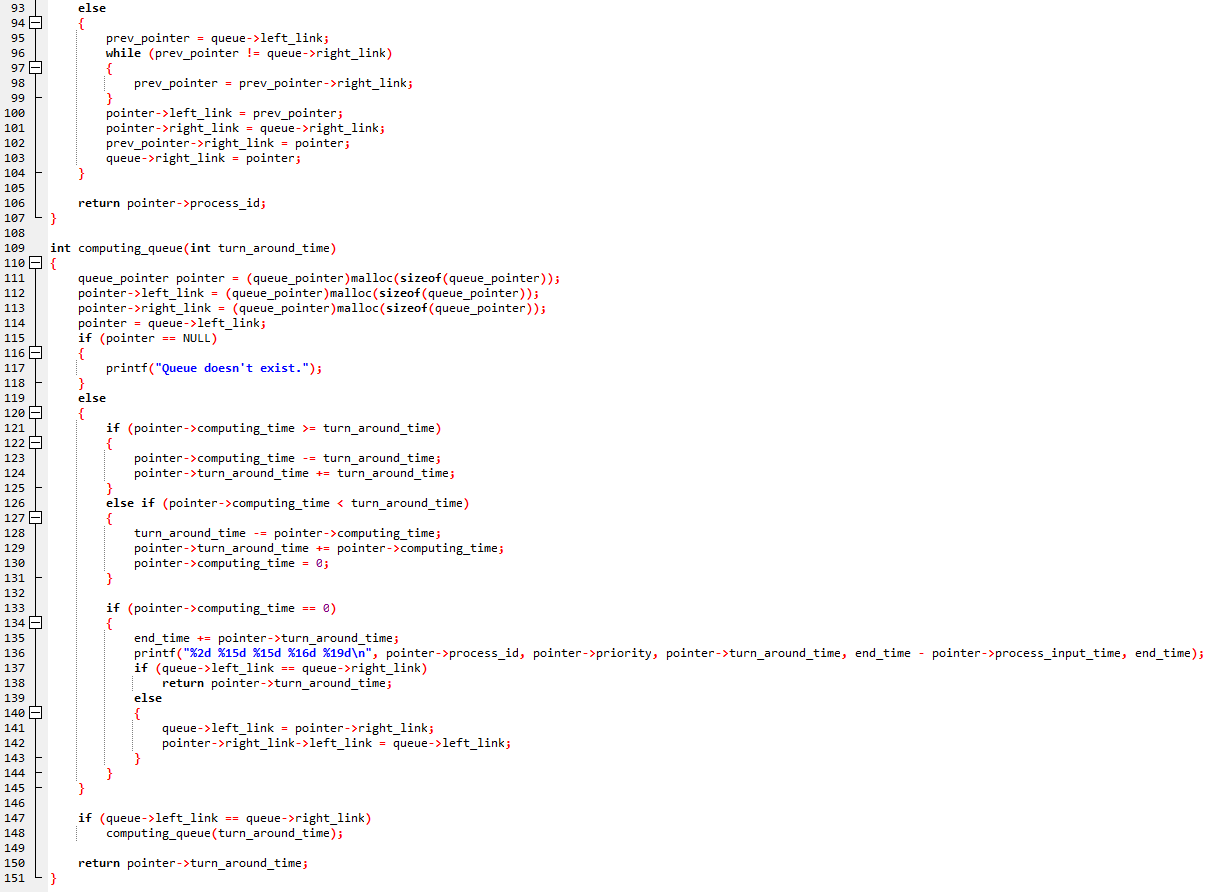
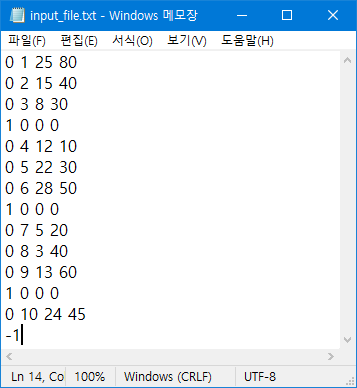
- Queue가 NULL인 경우 가장 앞에, 그렇지 않으면 가장 뒤에 프로세스를 삽입한다.

⑤ 프로세스 수행(computing\_queue)

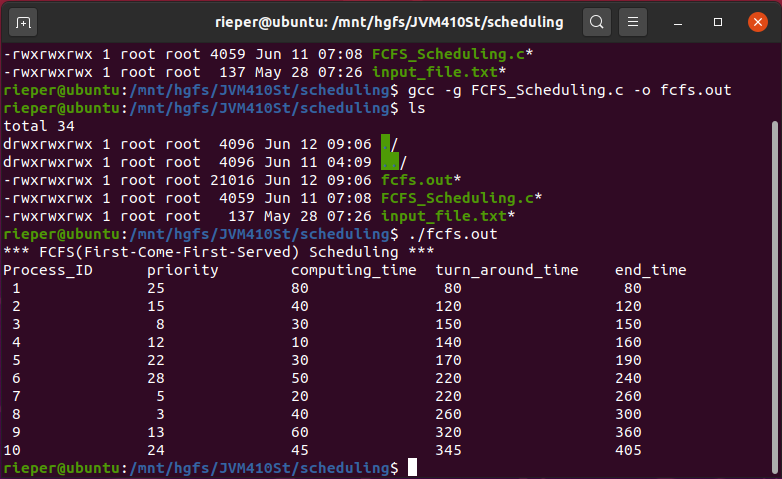
- Queue에 프로세스가 존재한다면, 가장 앞에 연결된 Queue(즉, 현재 실행 중인 프로세스)의 computing\_time을 time\_quantum(=20)만큼 감소시킨다. 이 때 그 값이 0보다 크다면 computing\_queue를 종료한다. computing\_time이 0보다 작거나 같으면(즉, 해당 프로세스의 수행이 끝나면), 해당 프로세스는 process\_id, priority, computing\_time, turn\_around\_time, end\_time을 출력하고, Queue에서 해당 프로세스를 제거한 뒤 computing\_queue를 종료한다.

- input\_file.txt에서 type -1을 입력 받으면 모든 프로세스의 수행이 종료될 때까지 해당 작업을 반복한다.

**4. 소스 코드(FCFS\_Scheduling.c)**

**5. 데모 결과 및 설명**

프로그램 실행에 이용된 입력 파일인 input\_file.txt의 내용이다.

ubuntu 환경에서 실행시켰을 때의 결과이다. VMWare를 이용한 환경이며, 폴더는 Windows와의 공유 폴더이다. gcc 컴파일러를 이용한 컴파일 과정 및 input\_file도 같이 존재하는 사실을 확인할 수 있다. 출력 결과를 다시 표로 나타내면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 프로세스  (Process\_ID) | 우선순위  (priority) | 수행 시간  (computing\_time) | Turn around time | 종료 시간  (end\_time) |
| 1 | 25 | 80 | 80 | 80 |
| 2 | 15 | 40 | 120 | 120 |
| 3 | 8 | 30 | 150 | 150 |
| 4 | 12 | 10 | 140 | 160 |
| 5 | 22 | 30 | 170 | 190 |
| 6 | 28 | 50 | 220 | 240 |
| 7 | 5 | 20 | 220 | 260 |
| 8 | 3 | 40 | 260 | 300 |
| 9 | 13 | 60 | 320 | 360 |
| 10 | 24 | 45 | 345 | 405 |

이러한 결과가 나오는 과정은 다음과 같다.

1. input\_file.txt에서 다른 type이 나오기 전까지의 type 0 데이터를 읽는다. 즉, 프로세스 ID가 각각 1, 2, 3인 데이터를 순서대로 읽을 것이고, 이는 순서대로 각각 1번, 2번, 3번 프로세스를 생성한다는 의미이다. 생성된 프로세스들은 순서대로 Queue에 들어가게 된다. 이때의 시간은 0이다. (t=0)
2. type이 1인 데이터를 읽으면, 프로그래밍 시 지정된 time quantum(=20)만큼 시간이 흐르게 되고, 실행 중인 프로세스의 computing\_time을 그만큼 감소시킨다. 이 때 1번 프로세스의 computing\_time은 80-20=60이 되고, 시간은 증가하여 t=20이 된다. 이 연산은 computing\_queue에 의해 실행된다.
3. 앞에서 type 1을 읽었다는 것은 곧 시간할당량이 끝났음을 의미하며, 다음 type 1이 등장하기 전까지의 프로세스들을 생성 및 Queue에 삽입해야 한다. 따라서 여기에서 3번, 4번, 5번 프로세스가 생성되어 Queue에 들어간다. 시간은 똑같이 t=20이고, 다른 요소도 변함이 없다.
4. 다시 type 1을 읽어 time quantum만큼 시간이 흐른다. 실행 중인 1번 프로세스의 computing\_time을 20만큼 감소시켜 60-20=40이 되고, 시간 t=40이 된다.
5. 다시 다른 type이 나오기 전까지의 type 0 데이터를 읽어 프로세스를 생성 및 Queue에 삽입한다. 여기에서 7번, 8번, 9번 프로세스가 생성되어 Queue에 들어간다.
6. 다시 type 1을 읽어 time quantum만큼 시간이 흐른다. 실행 중인 1번 프로세스의 computing\_time을 20만큼 감소시켜 40-20=20이 되고, 시간 t=60이 된다.
7. 다시 다른 type이 나오기 전의 type 0의 데이터를 읽는다. 10번 프로세스가 실행 및 스케줄링된다.
8. -1을 읽어 파일로부터의 입력을 종료하고, Queue의 프로세스들을 처리하기만 하면 된다. 여기에서 1번 프로세스가 종료되기까지의 computing\_time은 20이 남았는데, 시간 t를 20만큼 증가시켜 t=80으로 만들고 1번 프로세스의 computing\_time을 그 만큼 빼서 0을 만들어 종료시킨다. 1번 프로세스가 종료되면서 process\_id, priority, computing\_time, turn\_around\_time, end\_time을 출력하고, Queue에서 해당 프로세스를 제거한 뒤 computing\_queue를 종료한다. t=80에서 종료되었으므로 end\_time은 80이고 t=0일 때 Queue에 들어왔으므로 turn\_around\_time은 80-0=80이다. 나머지 값은 처음 파일에서 입력 받은 그대로이다.
9. 이제 2번 프로세스를 처리한다. 2번 프로세스의 computing\_time은 40인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=120이 된다. computing\_time이 0이 되었으므로 2번 프로세스도 process\_id, priority, computing\_time, turn\_around\_time, end\_time을 출력하고, Queue에서 해당 프로세스를 제거한 뒤 computing\_queue를 종료한다. t=120에서 종료되었으므로 end\_time=120이고, t=0일 때 Queue에 들어왔으므로 turn\_around\_time은 120-0=120이다.
10. 이제 3번 프로세스를 처리한다. 3번 프로세스의 computing\_time은 30인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=150이 된다. computing\_time이 0이 되었으므로 3번 프로세스도 위와 같은 과정을 반복한다. t=150에서 종료되었으므로 end\_time=150이고, t=0일 때 Queue에 들어왔으므로 turn\_around\_time은 150-0=150이다.
11. 이제 4번 프로세스를 처리한다. 4번 프로세스의 computing\_time은 10인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=160이 된다. computing\_time이 0이 되었으므로 4번 프로세스도 위와 같은 과정을 반복한다. t=160에서 종료되었으므로 end\_time=160이고, t=20일 때 Queue에 들어왔으므로 turn\_around\_time은 160-20=140이다.
12. 이제 5번 프로세스를 처리한다. 5번 프로세스의 computing\_time은 30인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=190이 된다. computing\_time이 0이 되었으므로 5번 프로세스도 위와 같은 과정을 반복한다. t=190에서 종료되었으므로 end\_time=190이고, t=20일 때 Queue에 들어왔으므로 turn\_around\_time은 190-20=170이다.
13. 이제 6번 프로세스를 처리한다. 6번 프로세스의 computing\_time은 50인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=240이 된다. computing\_time이 0이 되었으므로 6번 프로세스도 위와 같은 과정을 반복한다. t=240에서 종료되었으므로 end\_time=240이고, t=20일 때 Queue에 들어왔으므로 turn\_around\_time은 240-20=220이다.
14. 이제 7번 프로세스를 처리한다. 7번 프로세스의 computing\_time은 20인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=260이 된다. computing\_time이 0이 되었으므로 7번 프로세스도 위와 같은 과정을 반복한다. t=260에서 종료되었으므로 end\_time=260이고, t=40일 때 Queue에 들어왔으므로 turn\_around\_time은 260-40=220이다.
15. 이제 8번 프로세스를 처리한다. 8번 프로세스의 computing\_time은 40인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=300이 된다. computing\_time이 0이 되었으므로 8번 프로세스도 위와 같은 과정을 반복한다. t=300에서 종료되었으므로 end\_time=300이고, t=40일 때 Queue에 들어왔으므로 turn\_around\_time은 300-40=260이다.
16. 이제 9번 프로세스를 처리한다. 9번 프로세스의 computing\_time은 60인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=360이 된다. computing\_time이 0이 되었으므로 9번 프로세스도 위와 같은 과정을 반복한다. t=360에서 종료되었으므로 end\_time=360이고, t=40일 때 Queue에 들어왔으므로 turn\_around\_time은 360-40=320이다.
17. 이제 10번 프로세스를 처리한다. 10번 프로세스의 computing\_time은 45인데, 그것을 0으로 만들고, 그 만큼의 시간을 증가시킨다. 따라서 시간 t=405이 된다. computing\_time이 0이 되었으므로 10번 프로세스도 위와 같은 과정을 반복한다. t=405에서 종료되었으므로 end\_time=405이고, t=60일 때 Queue에 들어왔으므로 turn\_around\_time은 405-60=345이다. 모든 프로세스의 실행이 종료되었으므로 프로그램을 종료한다.

해당 과정을 설명한 그림을 보고서의 가장 마지막에 첨부한다.

**5. 분석 및 결론**

먼저 위의 과정에서 얻은 표를 바탕으로 정규화 된 반환시간(normalized turnaround time)을

구해야 한다. 다음은 해당 과정을 통해 얻은 표이다.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 프로세스 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  |
| 도착 시간 | 0 | 0 | 0 | 20 | 20 | 20 | 40 | 40 | 40 | 60 |  |
| 서비스 시간 | 80 | 40 | 30 | 10 | 30 | 50 | 20 | 40 | 60 | 45 |  |
| 종료 시각 | 80 | 120 | 150 | 160 | 190 | 240 | 260 | 300 | 360 | 405 | 평균 |
| 반환 시간 | 80 | 120 | 150 | 140 | 170 | 220 | 220 | 260 | 320 | 345 | 202.5 |
|  | 1.00 | 3.00 | 5.00 | 14.0 | 5.67 | 4.40 | 11.0 | 6.50 | 5.33 | 7.67 | 6.36 |

FCFS 알고리즘의 Normalized Turnaround Time은 약 6.36이 된다. 혼자서 팀을 구성하여 하나의 알고리즘만을 구현하여 실행하였기에 다른 스케줄링 알고리즘과 비교할 수는 없었지만, 수업 시간에 들은 내용을 토대로 미루어 보면 FCFS 방식보다 좋은 방법이 많이 있으므로 실제로 스케줄링 알고리즘을 이용한 프로그래밍을 한다면 FCFS 외의 다른 알고리즘을 이용하는 것이 바람직하리라고 생각된다.