**운영체제 실습과제 설명서**

라운드로빈(RoundRobin)

2019.12.20.

**P.P조 박희지**

**목 차**

**Ⅰ. 개발 도구** **3**

**Ⅱ. 구성3**

2.1전체 구성3

2.2 소스코드 구성4

**Ⅲ. 플로우차트 5**

3.1시스템 전체 플로우 차트5

3.2 ReadProcessInfo() 플로우차트6

3.3 Scheduling() 플로우차트 7

3.4 WriteExecutionResult() 플로우차트 8

**Ⅳ. 함수 기능 명세** **10**

4.1 ProcessClass10

4.1.1 Constructor10

4.1.2 Set 함수10

4.1.3 Get 함수10

4.2 RRScheduling11

4.2.1 Constructor11

4.2.2 Set 함수11

4.2.3 Get 함수11

4.3 그 외 함수12

4.3.1 ReadProcessInfo()12

4.3.2 Scheduling()12

4.3.3 WriteExecutionResult()13

4.3.4 Array 관련 함수13

**Ⅴ. 실행 결과 13**

5.1 cmd 창 출력 결과13

5.2 txt파일 출력 결과14

**Ⅵ. 소스코드16**

6.1 main.cpp16

6.2 header.h17

6.2.1 ProcessClass17

6.2.2 RRScheduling17

6.2.3 그 외 함수18

6.3 header.cpp18

6.3.1 ProcessClass18

6.3.2 RRScheduling19

6.3.3 ReadProcessInfo()20

6.3.4 Scheduling()21

6.3.5 WriteExecutionResult()22

6.3.6 Array 관련 함수22

**Ⅰ. 개발 도구**

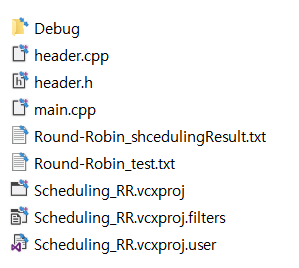
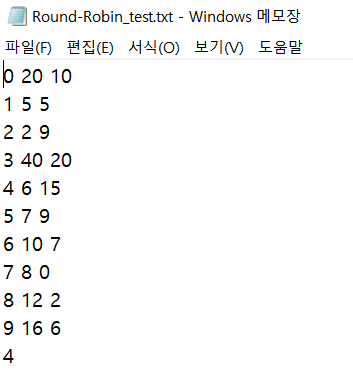
* 개발 언어: C++
* 개발 환경(IDE): Visual Studio
* 라이브러리: C++ STL 라이브러리(<queue>, <vector>, <algorithm> 사용)

**Ⅱ. 구성**

2.1. 전체 구성

아래의 [그림1]은 소스코드 프로젝트 내부 파일 목록으로, header.h라는 헤더파일에 라운드로빈에 필요한 클래스 및 함수들을 선언해주고, header.cpp에 클래스 내장 함수 및 그 외 함수들을 구분해서 작성하였다. 그리하여 main.cpp에서는 헤더파일에서 선언한 라운드로빈 함수를 main() 함수에서 사용하기만 하면 라운드로빈 스케줄링 알고리즘에 의한 시스템 스케줄링이 실행된다.

‘Round-Robin\_test.txt’는 프로세스의 정보가 담긴 txt파일이다. [그림2]를 살펴보면 1번째 컬럼은 프로세스 아이디(id), 2번째 컬럼은 작업시간(BurstTime), 그리고 3번째 컬럼은 도착시간(ArrivalTime)을 나타낸다. 그렇게 총 10개의 프로세스 정보들이 담겨있고 마지막 로우에는 한 프로세스가 작업을 할당받을 수 있는 시간, 타임 슬라이스(Quantum)를 담고 있다. 그리하여 라운드로빈을 실행하면 해당 txt파일 속 프로세스들을 라운드로빈 기법으로 스케줄링을 하는데, 스케줄링 정보를 출력한 txt파일이 ‘Round-Robin\_schedulingResult.txt’이다. 스케줄링 정보가 담긴 txt파일의 세부 내용은 “Ⅴ. 실행 결과” 목차에서 다시 언급이 되어 있다.

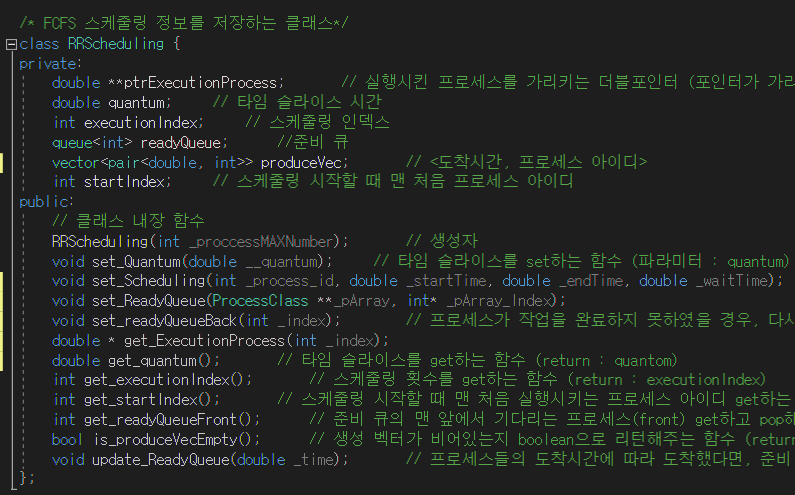
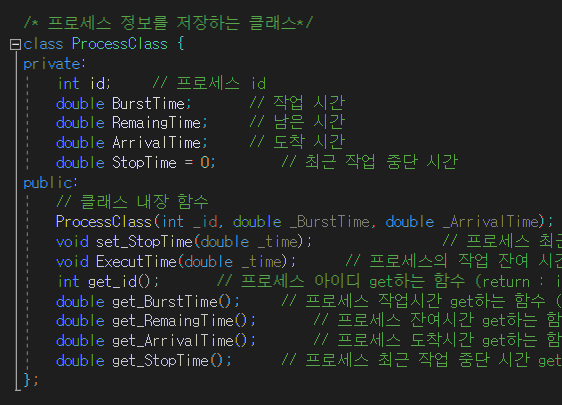
****

[그림1. Scheduling\_RR 프로젝트 파일 구성] [그림2. 프로세스 정보가 담긴 txt파일]

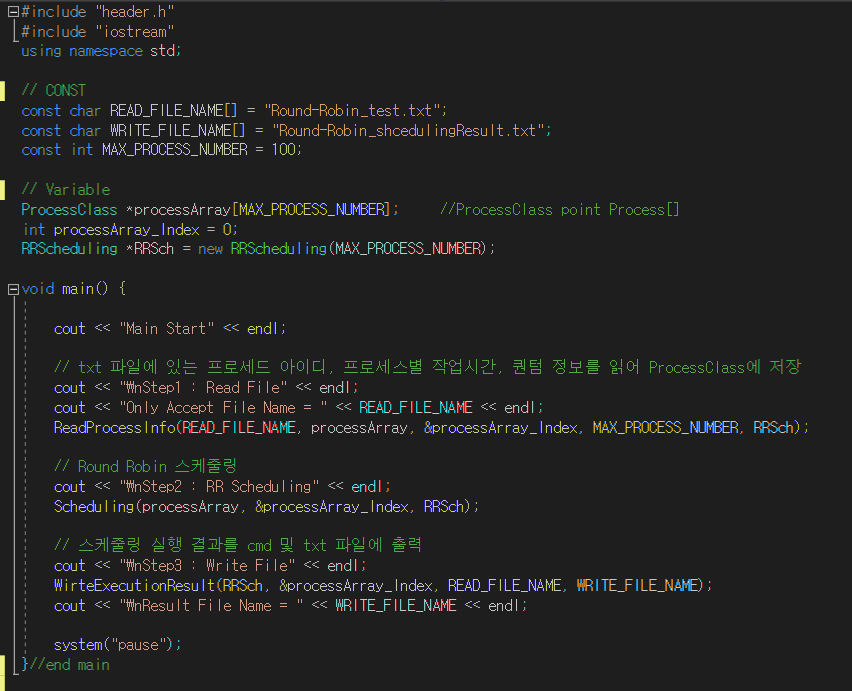
2.2. 소스코드 구성

헤더파일 header.h내에 프로세스 정보를 저장하는 ProcessClass, 스케줄링 정보를 저장하는 RRscheduling, 그리고 두 클래스의 객체를 파라미터로 사용하는 함수들이 선언되어 있다. 그래서 [그림4]를 보면, main.cpp에서 입출력 파일 이름, 프로세스 생성 최대 개수와 ProcessClass 및 RRscheduling 객체를 선언만 해주면, main()함수에서 header.h에서 선언한 함수에 파라미터로 이들을 넘겨주어 프로세스정보를 읽어 저장하고, 라운드로빈을 실행하는 모습을 확인할 수 있다. 여기서 객체들을 포인터형식(\*processArray, \*RRSch)으로 선언하여, 함수에는 객체의 주소값을 넘김으로써 참조에 의한 호출 방식(Call by reference)으로 객체에 접근할 수 있게 하였다.

또한 각 클래스들 안 변수들을 private으로 초기화함으로써, 클래스 외부에서 사용자의 직접 접근을 차단하고, 클래스 내장 함수를 public으로 선언함으로써, 함수 호출을 통해서만 자원에 접근 가능하도록 하는 구조로 클래스를 설계하였다.



[그림3. header.h에 선언된 ProcessClass와 RRscheduling 클래스]

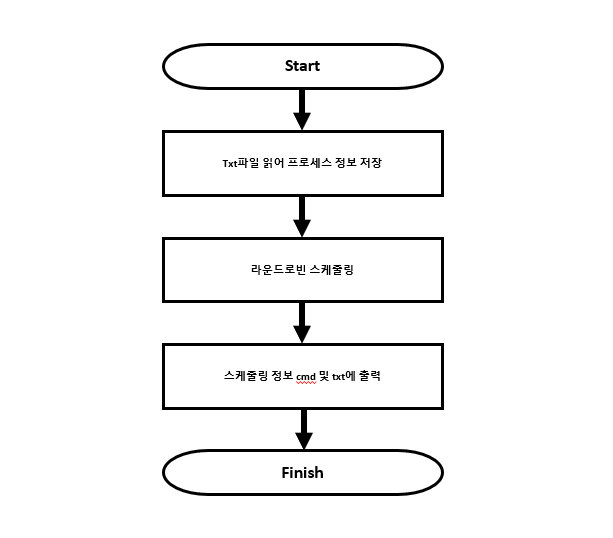
****

[그림4. main.cpp 소스코드]

**Ⅲ. 플로우차트**

3.1 시스템 전체 플로우차트

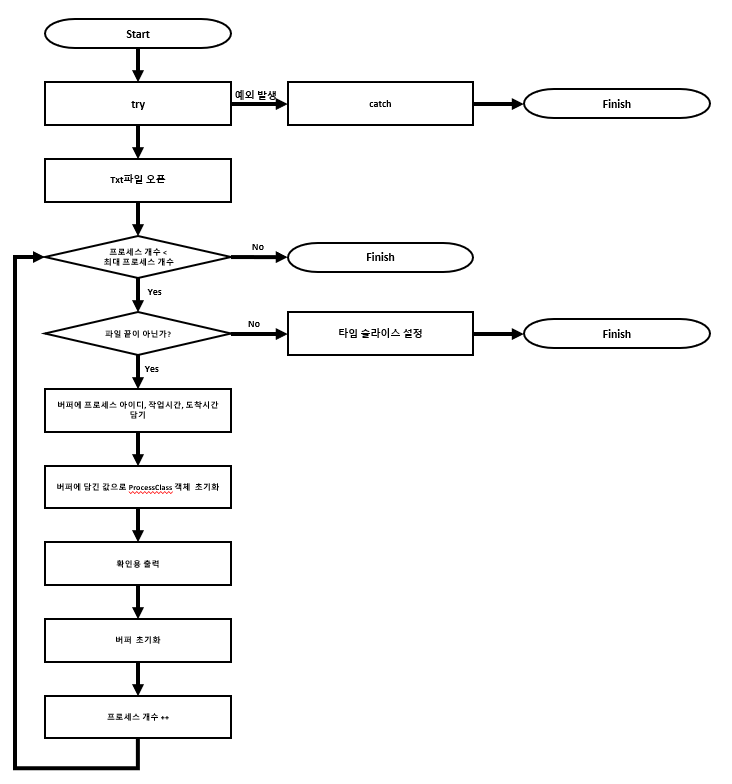
소스코드의 전체 플로우차트는 [그림5]와 같다. 아래와 같이 각 3단계를 차례대로‘ReadProcessInfo()’, ‘Scheduling()’, ‘WriteExecutionResult()’ 함수를 이용하여 실행시키므로 아래에 각 함수에 대한 플로우차트를 제시하였다.



[그림5. 라운드로빈 시스템 전체 플로우차트]

3.2 ReadProcessInfo() 플로우차트

프로세스 정보가 담긴 txt파일을 불러와 ProcessClass 객체에 프로세스들의 정보를 저장하게 해주는 함수로 플로우차트는 다음과 같다.



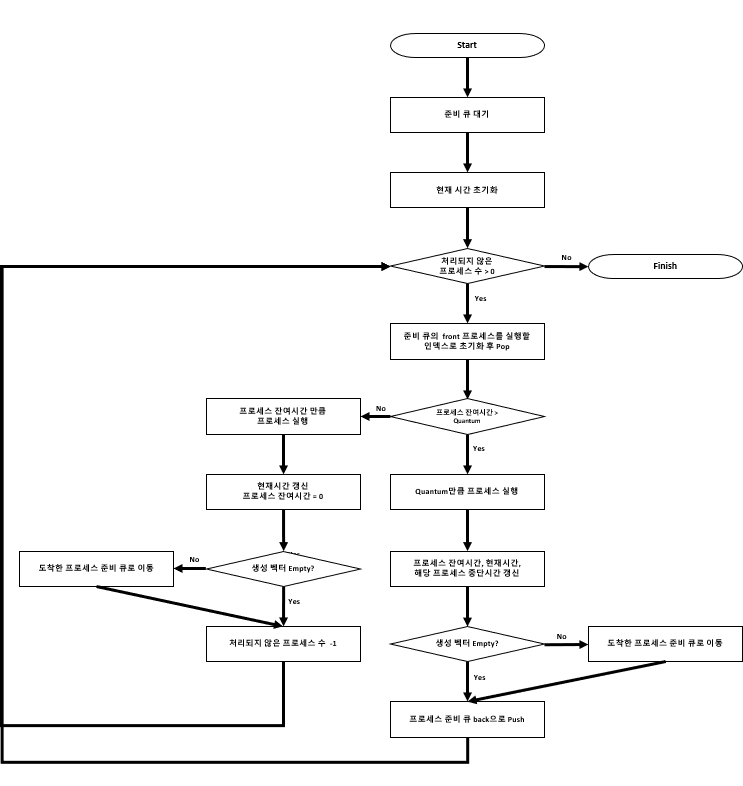
[그림6. ReadProcessInfo() 플로우차트]

3.3 Scheduling() 플로우차트

프로세스 정보가 담긴 ProcessClass 객체들을 불러와서 RRScheduling 객체의 생성단계의 벡터에 <도착시간, 프로세스 아이디> 짝으로 도착시간 기준으로 오름차순으로 정렬하여 저장 후, 벡터의 첫번째 원소의 프로세스 아이디를 준비 큐로 이동시켜 준비 큐를 대기시킨다. 이에 따라 준비 큐의 front 프로세스는 가장 먼저 도착한 프로세스이므로, 스케줄링 시작 전 현재 시간을 준비 큐의 front 프로세스의 도착시간으로 초기화시킨다.

그리고 프로세스를 실행시킬 때에 프로세스 아이디, 시작시간, 종료시간, 해당 프로세스의 대기시간 등의 스케줄링 정보를 RRScheduling 객체에 저장한다.

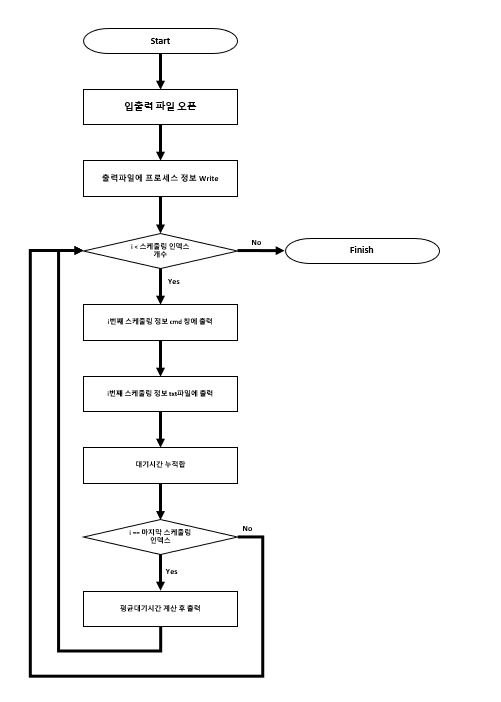
또한 프로세스 실행에 의해 현재 시간이 갱신함에 따라 생성 벡터에 있는 프로세스들이 도착하였다면 준비 큐로 이동시키면서 준비 큐를 갱신시켜준다.



[그림7. Scheduling() 플로우차트]

3.4 WriteExecutionResult() 플로우차트

WriteExecutionResult() 함수는 스케줄링 정보에는 RRScheduling 객체로부터 프로세스 아이디, 시작시간, 종료시간, 해당 프로세스의 대기시간을 get하여 프로세스 실행결과를 cmd 및 txt파일에 출력해주고, 스케줄링 정보가 가지는 대기시간들의 평균을 구하여 평균대기시간까지 출력해준다.



[그림8. WriteExecutionResult () 플로우차트]

**Ⅳ. 함수 기능 명세**

4.1 ProcessClass

4.1.1 Constructor

ProcessClass::ProcessClass(int \_id, double \_BurstTime, double \_ArrivalTime)

* 파라미터 : 프로세스 아이디(\_id), 작업시간(\_BurstTime), 도착시간(\_ArrivalTime)
* 기능 : 생성을 하면, ProcessClass 객체에 프로세스 아이디, 작업시간, 도착시간, 작업시간, 프로세스 잔여 시간, 최근 중단 시간을 set해준다. 잔여시간은 작업시간으로 초기화해주고, 최근 중단 시간은 도착시간으로 초기화 시켜준다.

4.1.2 Set 함수

void ProcessClass::ExecutTime(double \_time)

* 파라미터 : 스케줄링 작업 시간(\_time)
* 기능 : 해당 프로세스의 잔여 시간을 갱신해준다. (잔여시간 = 잔여시간 – 스케줄링 작업 시간)

void ProcessClass::set\_StopTime(double \_time)

* 파라미터 : 현재 시간(\_time)
* 기능 : 프로세스 최근 작업 중단 시간을 set하는 함수 (중단 시간 = 현재 시간)

4.1.3 Get 함수

int ProcessClass::get\_id() { return id; }

* 리턴 : 프로세스 아이디(id)
* 기능 : 프로세스의 아이디를 get해준다.

double ProcessClass::get\_BurstTime() { return BurstTime; }

* 리턴 : 프로세스의 작업시간(BurstTime)
* 기능 : 프로세스의 작업시간을 get해준다.

double ProcessClass::get\_RemaingTime() { return RemaingTime; }

* 리턴 : 프로세스의 잔여시간(RemaingTime)
* 기능 : 프로세스의 잔여시간을 get해준다.

double ProcessClass::get\_ArrivalTime() { return ArrivalTime; }

* 리턴 : 프로세스의 도착시간(ArrivalTime)
* 기능 : 프로세스의 도착시간을 get해준다.

double ProcessClass::get\_StopTime() { return StopTime; }

* 리턴 : 해당 프로세스의 최근 중단 시간(StopTime)
* 기능 : 해당 프로세스의 최근 중단 시간을 get해준다.

4.2 RRScheduling

4.2.1 Constructor

RRScheduling::RRScheduling(int \_processMAXNumber)

* 파라미터 : 생성 가능한 프로세스 최대 개수(\_processMAXNumber)
* 기능 : 생성 가능한 프로세스 최대 개수만큼 더블포인터를 선언해주고, 스케줄링 정보(프로세스 아이디, 시작시간, 종료시간, 해당 프로세스의 대기시간)를 담을 포인터배열을[\_\_processMAXNumber][4] 크기로 할당해준다. 그리고 스케줄링 인덱스를 0으로 초기화시킨다.

4.2.2 Set 함수

void RRScheduling::set\_Quantum(double \_quantum) { quantum = \_quantum; }

* 파라미터 : 타임 슬라이스(\_quantum)
* 기능 : 타임슬라이스를 set해준다.

void RRScheduling::set\_Scheduling(int \_process\_id, double \_startTime, double \_endTime,   
 double \_waitTime)

* 파라미터 : 프로세스 아이디(\_process\_id), 시작시간(\_startTime), 종료시간(\_endTime), 대기시간(\_waitTime)
* 기능 : 스케줄링 정보를 저장하는 포인터배열 ptrExecutionProcess에 프로세스 아이디, 시작시간, 종료시간, 대기시간을 저장해주고, 스케줄링 인덱스를 ++하여 다음 인덱스로 가리키게 한다.

void RRScheduling::set\_ReadyQueue(ProcessClass \*\*\_pArray, int\* \_pArray\_Index)

* 파라미터 : 프로세스정보 객체Array(\*\*\_pArray), 프로세스 개수(\_pArray\_Index)
* 기능 : 스케줄링 시작 전, 준비 큐 대기시키는(set) 함수, 생성 단계의 프로세스들의 도착시간 및 프로세스 아이디를 pair vector<도착시간, 프로세스 아이디>에 저장하고, 도착시간 기준으로 오름차순으로 vector를 정렬한다. 이후 준비 큐에 맨 처음의 생성 단계의 프로세스 아이디를 startIndex로 초기화해주고, push 후 vector에서 지워준다.

void RRScheduling::set\_readyQueueBack(int \_index) { readyQueue.push(\_index); }

* 파라미터 : 프로세스 인덱스(\_index)
* 기능 : 프로세스가 작업을 완료하지 못하였을 경우, 다시 준비 큐의 back으로 push해준다.

void RRScheduling::update\_ReadyQueue(double \_time)

* 파라미터 : 현재 시간(\_time)
* 기능 : vector 안 생성 단계의 프로세스들 중, 도착시간이 현재 시간보다 빠르다면(프로세스가 도착했다면), 준비 큐로 프로세스 아이디를 push해준다.

4.2.3 Get 함수

double \* RRScheduling::get\_ExecutionProcess(int \_index) { return ptrExecutionProcess[\_index]; }

* 파라미터 : 스케줄링 인덱스(\_index)
* 리턴 : 특정 스케줄링 인덱스\_index)의 스케줄링 정보(ptrExecutionProcess[\_index])
* 기능 : 특정 스케줄링 인덱스(\_index)의 스케줄링 정보를 get해준다.
* 사용 예시   
  get\_ExecutionProcess(i)[0]; : i번째 스케줄링 인덱스의 프로세스 아이디 get  
  get\_ExecutionProcess(i)[1]; : i번째 스케줄링 인덱스의 스케줄링 시작시간 get  
  get\_ExecutionProcess(i)[2]; : i번째 스케줄링 인덱스의 스케줄링 종료시간 get  
  get\_ExecutionProcess(i)[3]; : i번째 스케줄링 인덱스의 해당 프로세스 대기시간 get

int RRScheduling::get\_executionIndex() { return executionIndex; }

* 리턴 : 스케줄링 인덱스(executionIndex) 🡸 스케줄링 횟수
* 기능 : 스케줄링 횟수(인덱스)를 get해준다.

double RRScheduling::get\_quantum() { return quantum; }

* 리턴 : 타임 슬라이스(quantum)
* 기능 : 타임 슬라이스를 get해준다.

int RRScheduling::get\_startIndex() { return startIndex; }

* 리턴 : 처음으로 실행시킬 프로세스 아이디(startIndex)
* 기능 : 스케줄링 시작할 때 제일 처음으로 실행시킬 프로세스 아이디(인덱스)를 get해준다.

int RRScheduling::get\_readyQueueFront() { return readyQueue.front(); }

* 리턴 : 준비 큐의 front 프로세스 아이디(readyQueue.front())
* 기능 : 준비 큐의 맨 앞에서 기다리는 프로세스(front) get하고 pop해준다.

bool RRScheduling::is\_produceVecEmpty() { return produceVec.empty(); }

* 리턴 : 생성 벡터가 비어있는지에 대한 상태 boolean(produceVec.empty())
* 기능 : 생성 벡터가 비어있는지 boolean으로 리턴해준다.

4.3 그 외 함수

4.3.1 ReadProcessInfo()

void ReadProcessInfo(const char \*\_FileName, ProcessClass \*\*\_processArray,   
 int \*\_proceeArray\_Index, const int MAX\_PROCESS\_NUMBER, RRScheduling \*\_RRSch)

* 파라미터 : READ용 txt파일 이름(\*\_FileName), 프로세스정보 객체Array(\*\*\_processArray), 총 프로세스 개수(\*\_proceeArray\_Index), 생성 가능한 프로세스 최대 개수(MAX\_PROCESS\_NUMBER), 스케줄링 정보 객체(\*\_RRSch)
* 기능 : txt파일을 불러와 프로세스 정보를 읽어와 \*\*\_processArray에 저장한다.

4.3.2 Scheduling()

void Scheduling(ProcessClass \*\*\_processArray, int\*\_processArray\_Index, RRScheduling \*\_RRSch)

* 파라미터 : 프로세스정보 객체Array(\*\*\_processArray), 총 프로세스 개수(\*\_proceeArray\_Index), 스케줄링 정보 객체(\*\_RRSch)
* 기능 : 라운드로빈 스케줄링을 실행한다.

4.3.3 WriteExecutionResult()

void WriteExecutionResult(RRScheduling \*\_RRSch, int\* \_processArray\_Index,  
 const char \*\_READ\_FILE\_NAME, const char \*\_WRITE\_FILE\_NAME)

* 파라미터 : 스케줄링정보 객체(\*\_RRSch), 총 프로세스 개수(\*\_processArray\_Index), READ용 txt파일 이름(\*\_READ\_FILE\_NAME), WRITE용 txt파일 이름(\*\_WRITE\_FILE\_NAME)
* 기능 : 스케줄링 실행결과를 cmd 창과 새로운 txt파일로 출력해준다.

4.3.4 Array 관련 함수

void showCharArray(char \*\_showchar, int \_charsize)

* 파라미터 : 출력할 CharArray(\*\_showchar), Array 크기(\_charsize)
* 기능 : Array를 출력해준다. (개발 시 값이 잘 담겼는지 확인용으로 만든 함수)

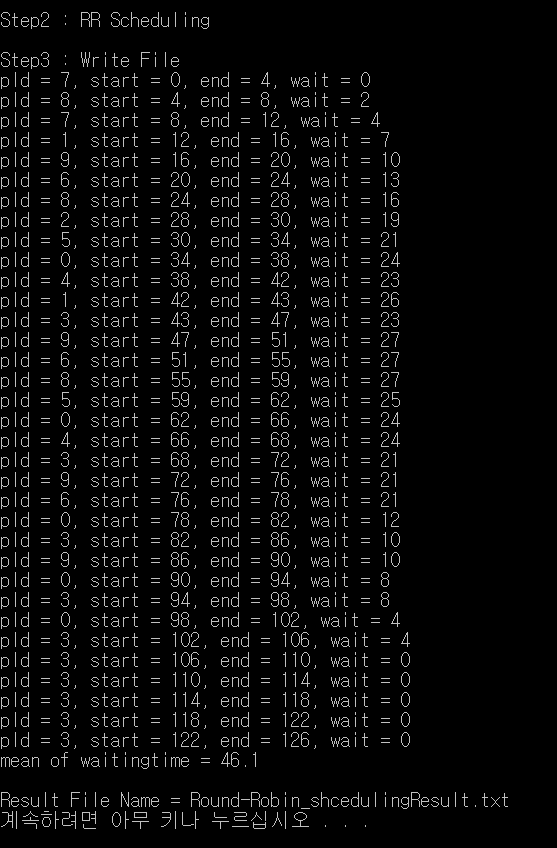
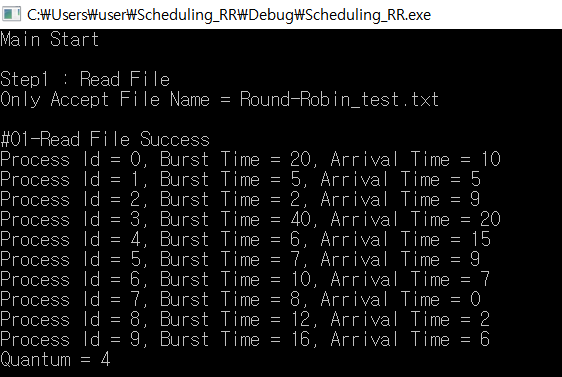
void resetCharArray(char \*\_resetchar, int \_charsize)

* 파라미터 : 초기화할 CharArray(\*\_showchar), Array 크기(\_charsize)
* 기능 : Array를 NULL로 초기화해준다.

**Ⅴ. 실행 결과**

5.1 cmd 창 출력 결과

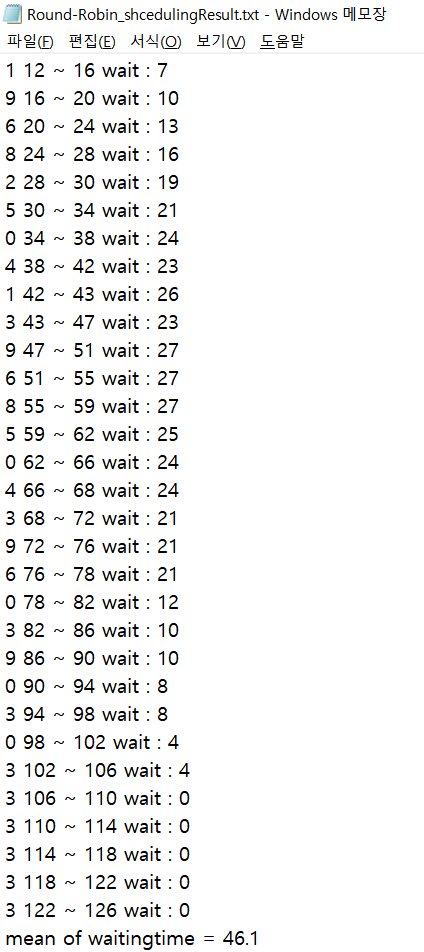
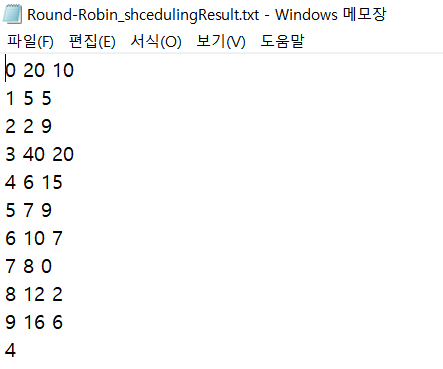
WriteExecutionResult() 함수를 실행시켜 RRScheduling 객체로부터 프로세스 아이디, 시작시간, 종료시간, 해당 프로세스의 대기시간 등의 스케줄링 정보들을 get하여 프로세스 실행결과를 cmd 창에 출력한 결과는 아래 [그림9], [그림10]과 같은 결과를 볼 수 있다.



[그림9. 10개의 프로세스 정보가 출력된 cmd] [그림10. 스케줄링 정보와 평균대기시간이 출력된 cmd]

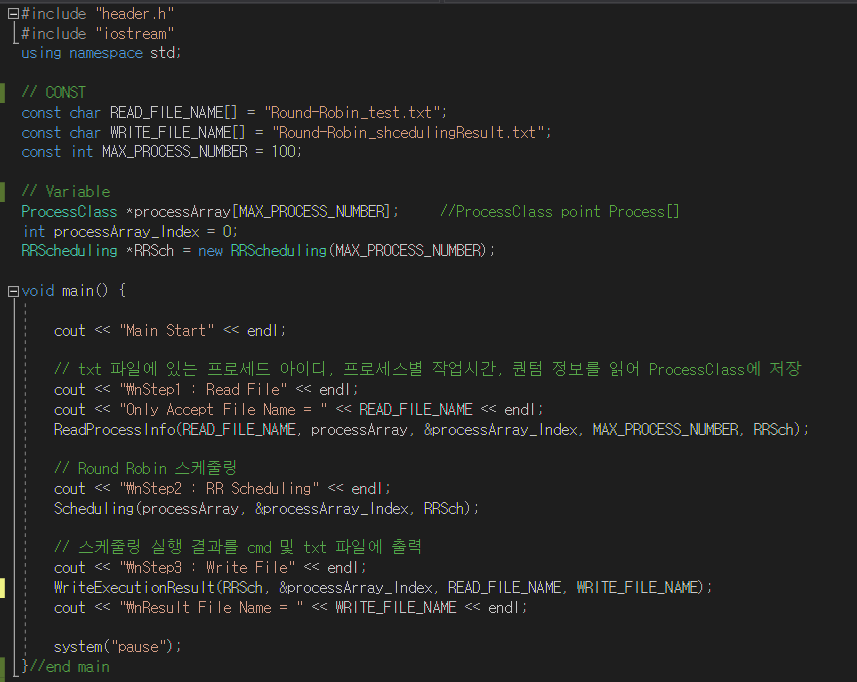
5.2 txt파일 출력 결과

WriteExecutionResult() 함수를 실행시켜 RRScheduling 객체로부터 프로세스 아이디, 시작시간, 종료시간, 해당 프로세스의 대기시간 등의 스케줄링 정보들을 get하여 프로세스 실행결과를 txt파일 “Round-Robin\_shcedulingResult.txt”에 출력한 결과는 아래 [그림11], [그림12]과 같은 결과를 가진다.



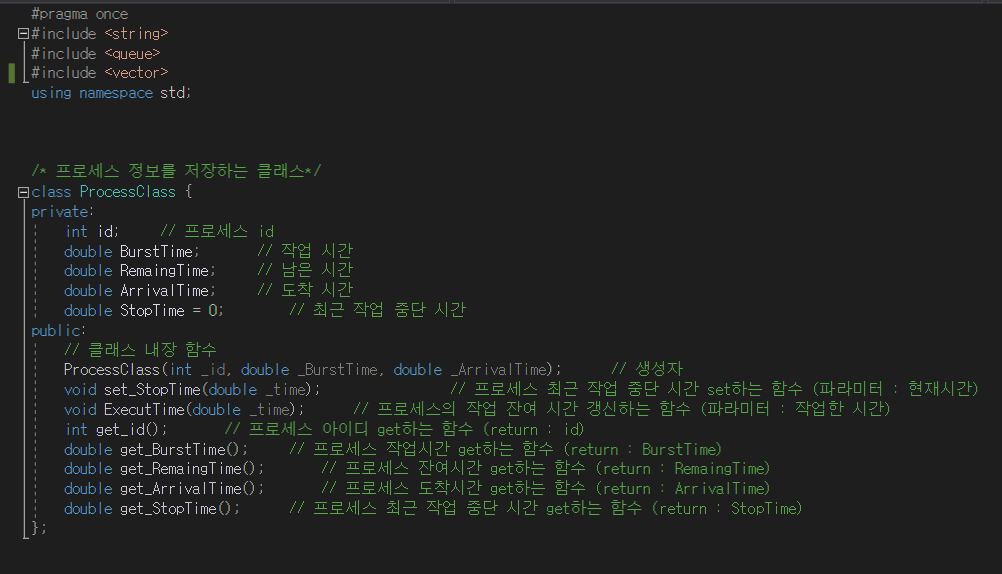
[그림11. 10개의 프로세스 정보가 출력된 txt파일] [그림12. 스케줄링 정보와 평균대기시간이 출력된 txt파일]

**Ⅵ. 소스코드**

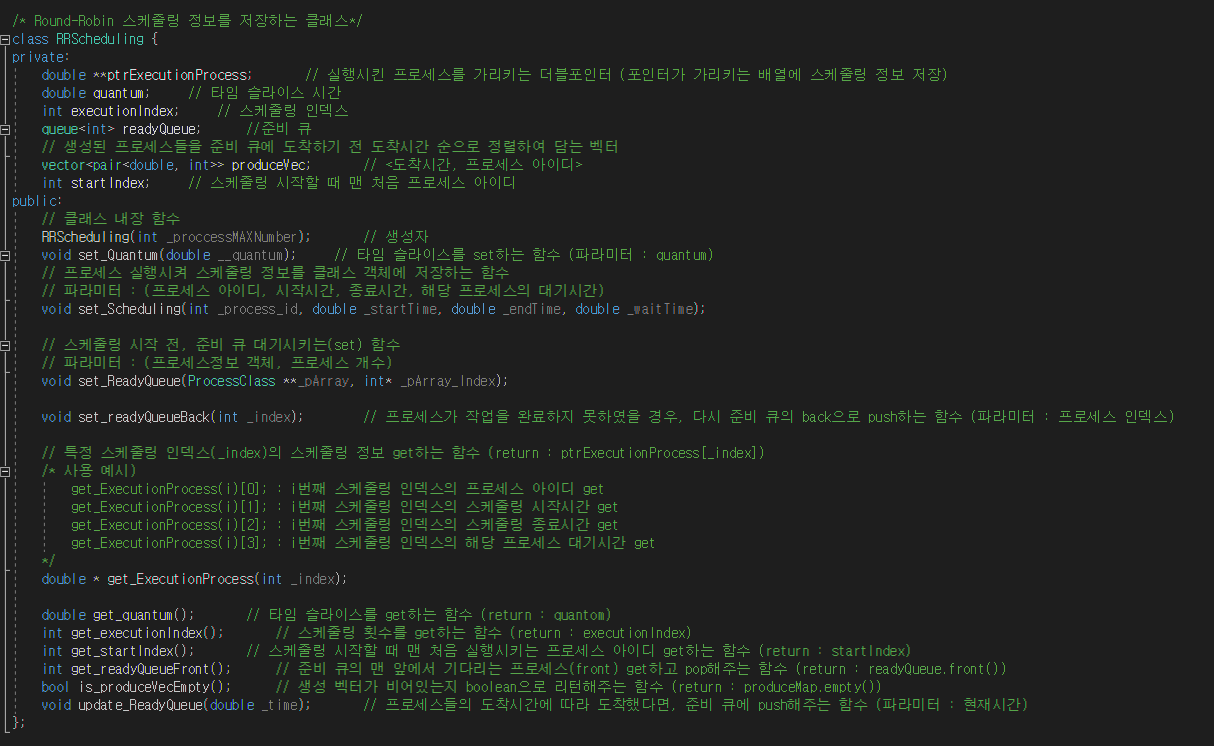
6.1 main.cpp 

6.2 header.h

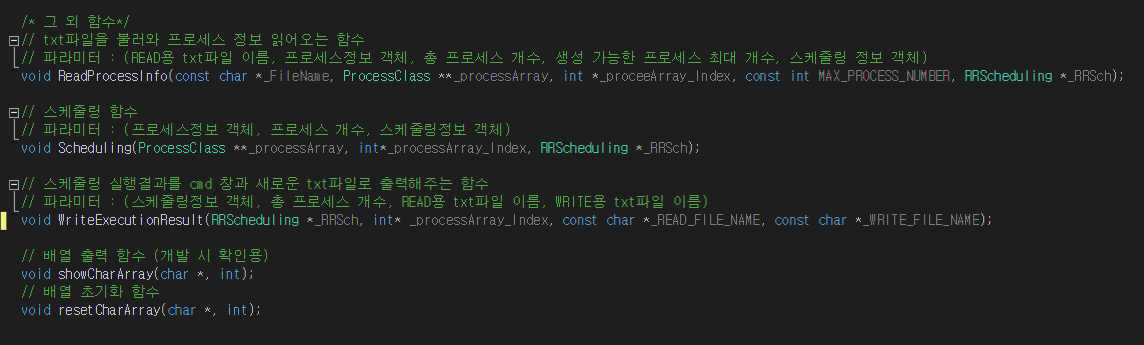
6.2.1 ProcessClass



6.2.2 RRScheduling

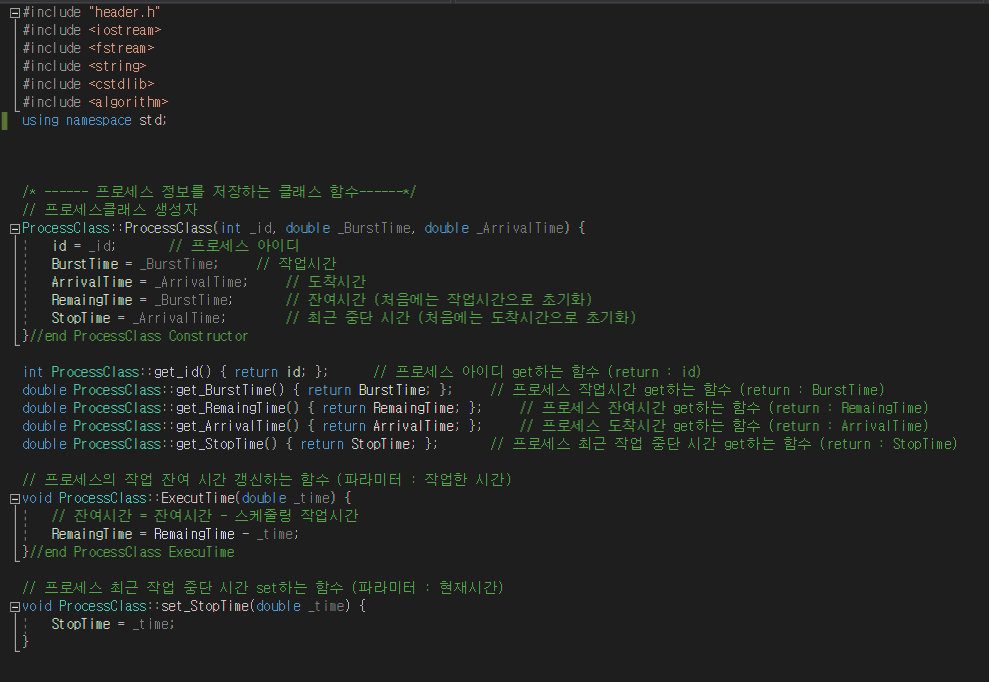


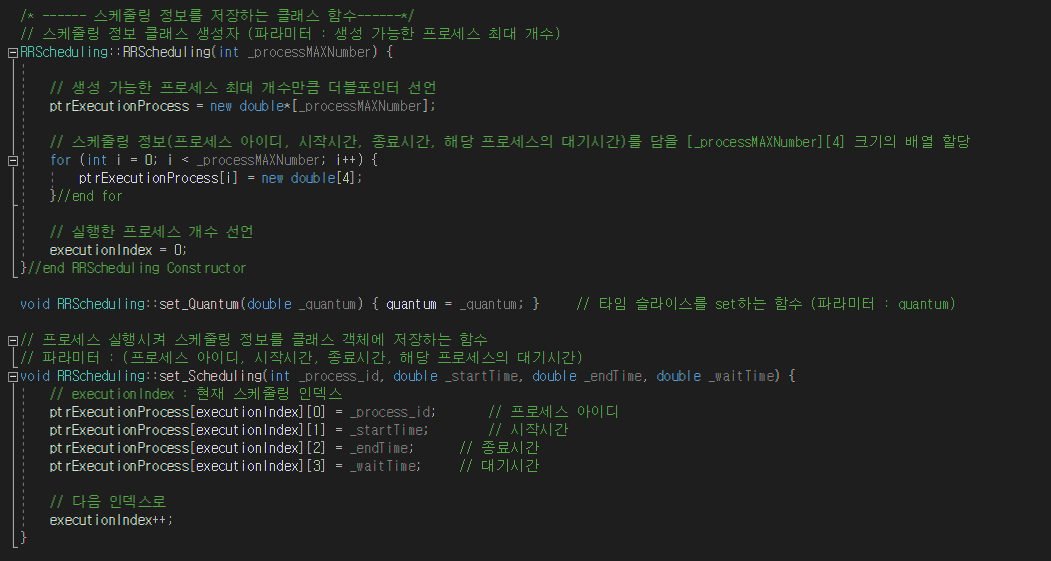
6.2.3 그 외 함수

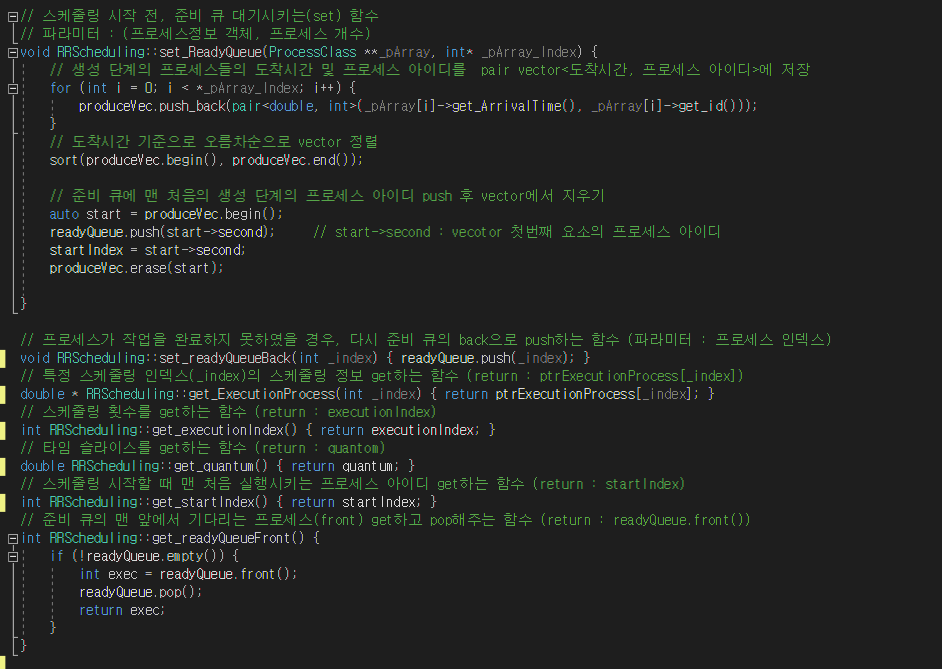
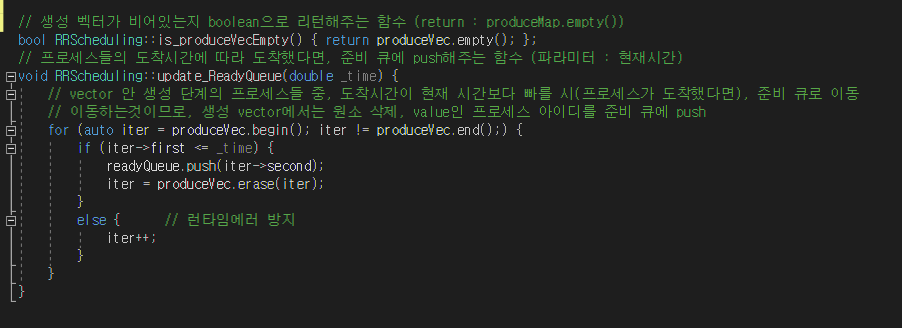


6.3 header.cpp

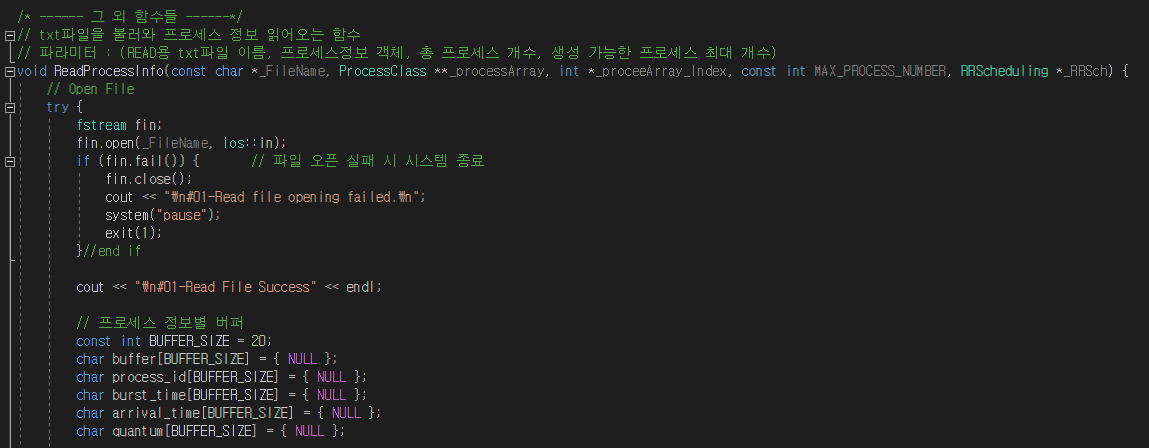
6.3.1 ProcessClass

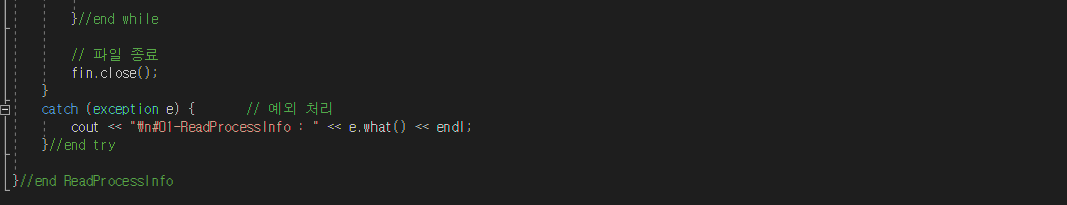
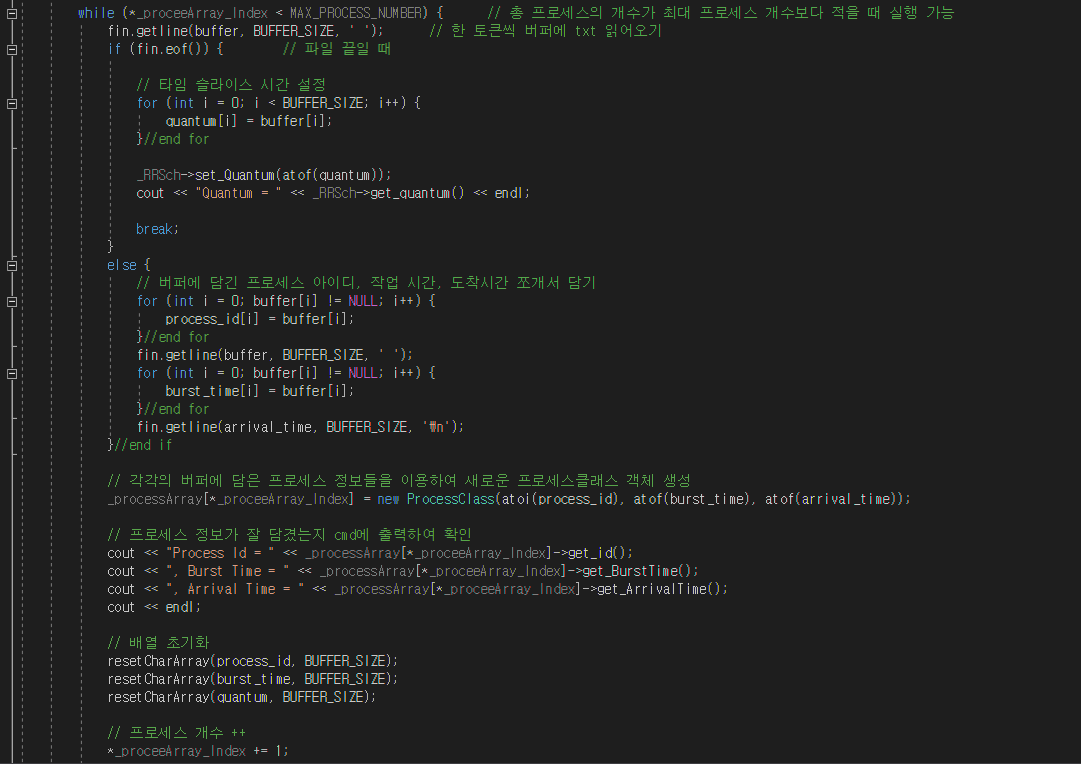


6.3.2 RRScheduling 

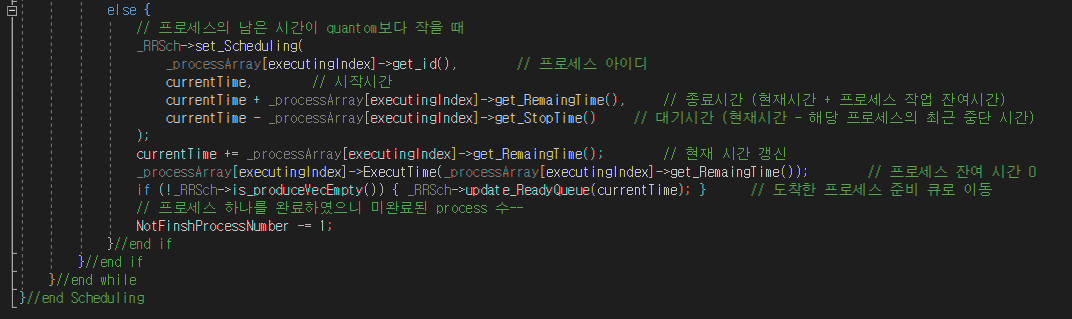
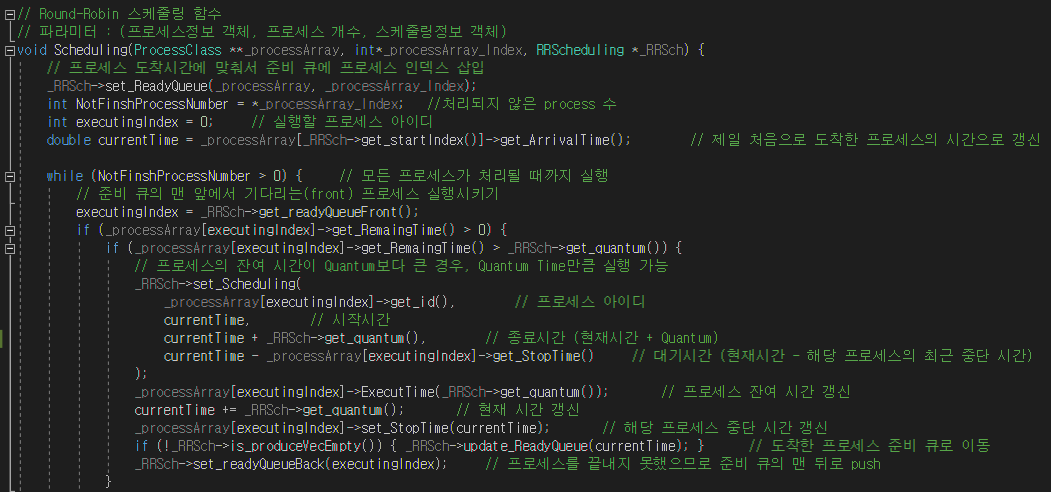
 

6.3.3 ReadProcessInfo()

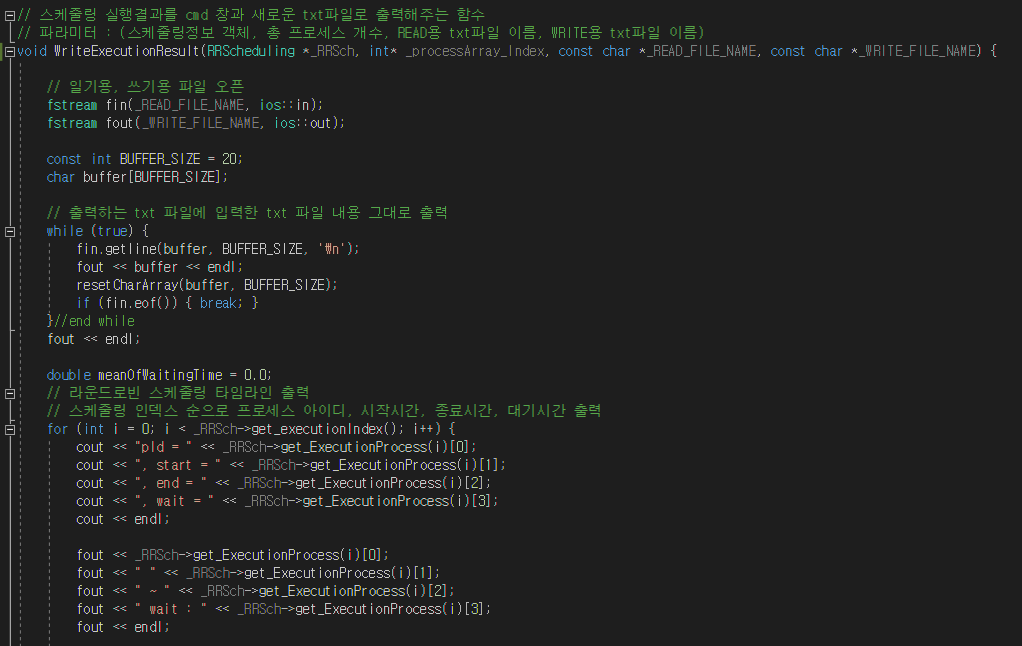


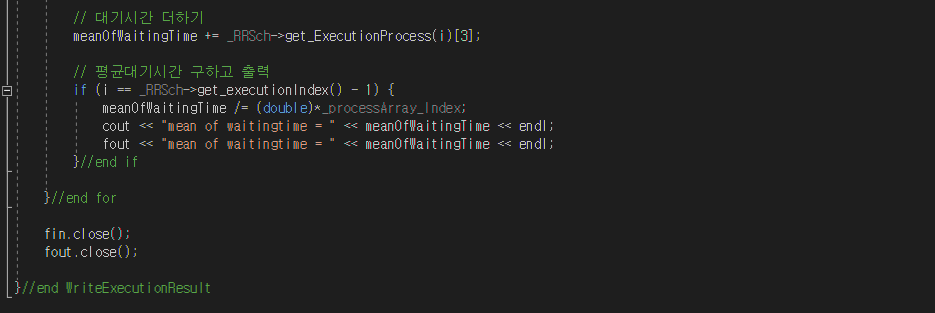


6.3.4 Scheduling()



6.3.5 WriteExecutionResult()





6.3.6 Array 관련 함수

