**소프트웨어 공학**

**유민수 교수님**

**컴퓨터공학부 컴퓨터전공**

**2013011424**

**성예닮**

**Project 1**

**테스트 문서**

**(Test Document)**

**목차**

**1. 개발 환경 및 테스트 환경**

**2. 사용 언어 및 IDE, 기타 툴**

**3. 알고리즘**

**4. 컴파일 및 테스트 방법**

**5. 테스트**

**1. 개발 환경 및 테스트 환경**

Windows 10 Home Edition Korea에서 코드 작성 및 테스트를 수행하였다.

**2. 사용 언어 및 IDE, 기타 툴**

중심 알고리즘 및 GUI 환경 제공은 Python 3.4.3 32bit 버전과 XML 1.0을 사용하였다.

주 에디터로는 github의 Atom io를 사용하였고, 디버깅을 위한 IDE로써 JetBrains PyCharm Community Edition 5.0.1를 사용하였다. 해당 에디터와 IDE에서는 라인 개행 및 파일 인코딩을 지정할 수 있고, 유닉스 계열 (맥 제외)과의 호환성을 갖추기 위해 라인 개행은 LF형식으로 처리하였고, 파일 인코딩은 UTF-8로 지정하여 안전한 한글과 영어 입력 및 출력이 가능하다.

사용자를 위한 GUI환경을 제공하기 위해 PyQt4와 Qt Designer을 사용하였습니다.

추가로, 프로젝트 디버깅 시에는 JetBrains PyCharm Community Edition 5.0.1을, 파일 하나를 디버깅 할 때에는 Wing IDE 101 5.1에서 디버깅 하였습니다.

**3. 알고리즘**

스케줄링 알고리즘은 RM, EDF, USER 세가지이다.

RM (Rate Monotonic) 알고리즘

- 주기적인 작업들만 고려하며, 주기가 짧은 순으로 스케줄링 한다.

- 대표적인 정적 스케줄링 방식이다.

ex) 같은 시각 (12)에 도착한 task1과 task2의 주기가 각각 10, 20이라면, task1을 먼저 수행한다.

EDF (Earliest Deadline First) 알고리즘

- 주기적, 비 주기적 작업들을 모두 고려하며, 마감 한계가 이른 순으로 스케줄링 한다.

- 대표적인 동적 스케줄링 방식이다.

ex) 같은 시각 (12)에 도착한 task1과 task2의 마감 한계가 각각 20, 30이라면, task1을 먼저 수행한다.

USER (사용자 우선순위) 알고리즘

- 본 프로젝트에서는 '사용자'를 '개발자'와 동일 인물로 보고, 개발자 본인이 사용자라고 생각하며 사용자 우선순위를 지정하였다.

- 본 프로젝트의 사용자는 작업이 생성되어 시작되어야 하는 시점에서만 작업이 수행되길 원한다. 해당 작업이 시작되어야 하는 시기를 놓치게 되면, 작업을 수행하지 않아야 한다.

- 시작 시간이 같다면 마감 한계가 이른 작업을 수행한다.

- 따라서 해당 스케줄링 알고리즘의 경우, 작업량이 많다면 불리한 알고리즘이다.

ex) 같은 시각 (12)에 도착한 task1과 task2의 마감 한계가 각각 20, 30이라면 task1만 수행한다.

**4. 컴파일 및 테스트 방법**

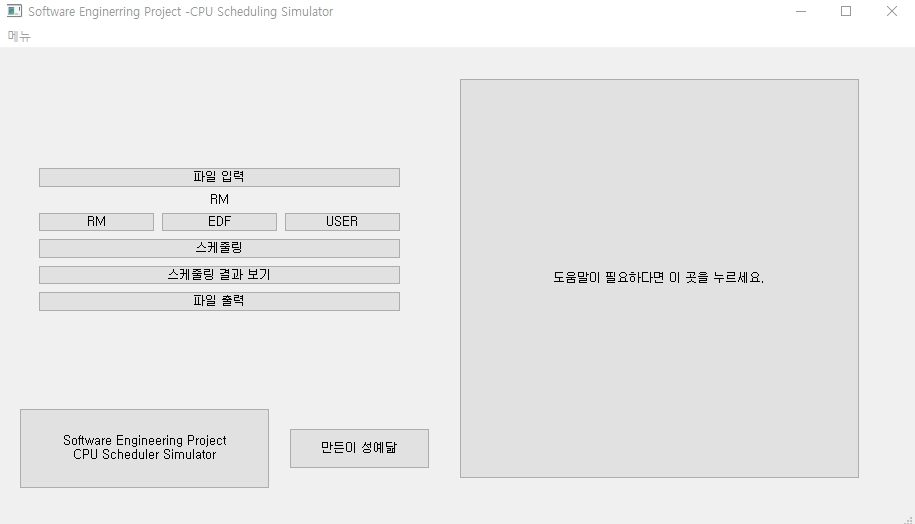
- 프로젝트 제출 시 포함된 압축 파일을 푼다.

- input.txt, main.py, Processing.py, Scheduler.py, Visualize.py, ui.ui는 모두 같은 디렉토리에 위치해야 한다.

- IDE에 프로젝트화 해서 모두 한 프로젝트 안에 위치시킨다.

- main.py를 컴파일 한 후 실행한다.

- 실행하게 되면 다음과 같은 창이 나타난다

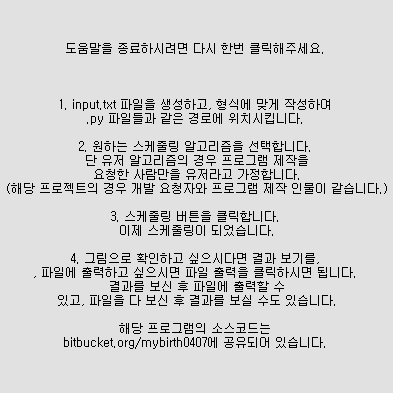


- 파일 입력-> 알고리즘 선택-> 스케줄링-> 스케줄링 결과 보기-> 파일 출력 순으로 사용한다. 단, 스케줄링 결과 보기와 파일 출력의 경우 어느 것이 먼저 선행되어도 프로그램 사용 시 지장이 전혀 없다. 스케줄링 과정까지는 위의 선행 순서를 따라야만 한다.

- 알고리즘을 선택하게 되면 다음과 같이 선택한 알고리즘이 표시된다.

C:\Users\Yedarm Seong\AppData\Local\Microsoft\Windows\INetCache\Content.Word\K-002.jpg

- 사용법이 기억이 나지 않는다면, 다음과 같은 버튼을 클릭하고, 도움말을 확인한다.



**5. 테스트**

- 테스트 환경은 Windows 10 Home Edition Korea 운영체제와 JetBrains PyCharm Community Edition 5.0.1이다.

- 테스트를 위한 input.txt 파일은 다음과 같다.

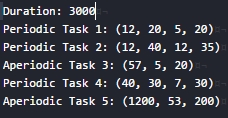
Duration: 3000  
Periodic Task 1: (12, 20, 5, 20)  
Periodic Task 2: (12, 40, 12, 35)  
Aperiodic Task 3: (57, 5, 20)  
Periodic Task 4: (40, 30, 7, 30)  
Aperiodic Task 5: (1200, 53, 200)

input.txt 파일을 간단히 수정하면 사용자가 원하는 작업을 생성해낼 수 있다.

**- RM 알고리즘**

앞 장의 도움말에 따라 테스트를 진행하였다.

input.txt 내용

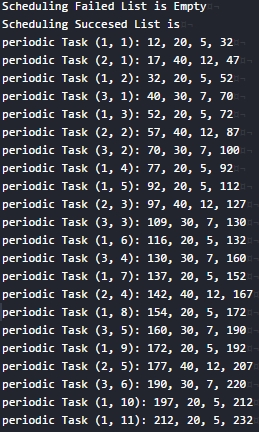


RM 알고리즘을 선택하였다.



스케줄링 버튼을 누른 후, 파일을 확인하였다.

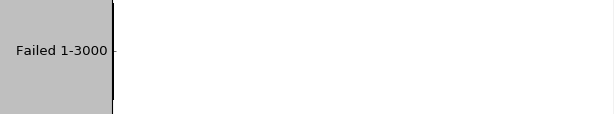
result.txt

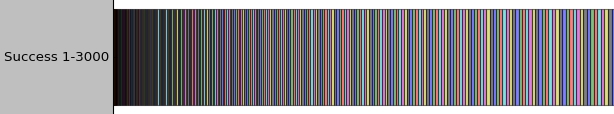


위와 같은 형태로 파일이 생성된다.

파일이 매우 길기 때문에 이하는 생략하였다.

스케줄링 결과 보기를 클릭하여 GUI환경으로 확인해보았다.





위에서 확인 할 수 있었듯, 입력한 duration동안 Failed 된 작업들이 없음을 확인할 수 있다.

RM 알고리즘의 경우, 주기적인 작업만 고려하기 때문에, 마감 시한을 적당히 넉넉하게 잡는다면, 최대한으로 많은 수행을 할 수 있다는 것을 알 수 있다.

스케줄링 과정을 살펴보면, input.txt에 의해 작업들이 다음과 같이 생성된다.

Task (1, 1): 12, 20, 5, 32  
Task (2, 1): 17, 40, 12, 47  
Task (1, 2): 32, 20, 5, 52  
Task (3, 1): 40, 30, 7, 70  
Task (1, 3): 52, 20, 5, 72  
Task (2, 2): 57, 40, 12, 87

해당 작업들의 경우, 모두 시작시간과 종료시간이 길지 않고, 마감 한계가 길어 서로 경쟁이 일어나서 몇 회 밀리게 되더라도 충분히 모두 스케줄링이 가능하다.

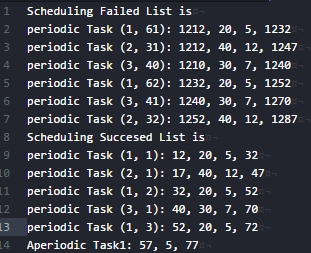
(Task 2의 1번 인스턴스의 시작시간이 17이 된 이유는, Task 1의 1번 인스턴스와 경쟁하였고, 주기가 길어 졌기 때문에 Task1을 먼저 실행한 후의 시작시간이기 때문이다..

**- EDF 알고리즘**

input.txt 파일은 앞선 RM 알고리즘과 동일하다.

C:\Users\Yedarm Seong\AppData\Local\Microsoft\Windows\INetCache\Content.Word\K-011.jpg

스케줄링 버튼을 누른 후, 출력 파일을 확인하였다.



마감 한계를 기준으로 스케줄링 하고, 비 주기적 알고리즘을 고려하였기 때문에 스케줄링 되지 않은 작업들이 존재한다.

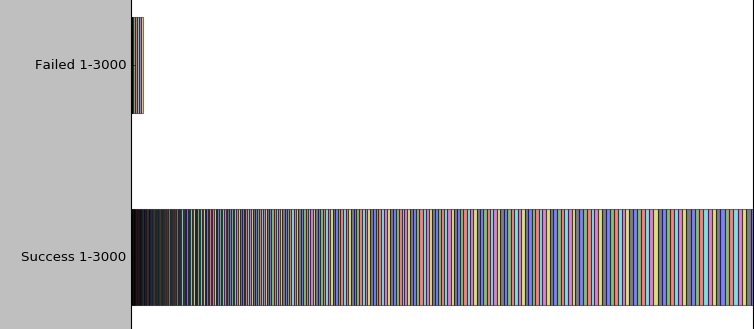
스케줄링이 실패한 작업들의 시간대를 보면, 다음과 같은 작업들이 처리되어야 함을 알 수 있습니다.

periodic Task (1, 60): 1196, 20, 5, 1212  
Aperiodic Task2: 1201, 53, 1400   
periodic Task (3, 40): 1210, 30, 7, 1240  
periodic Task (1, 61): 1212, 20, 5, 1232  
periodic Task (2, 31): 1212, 40, 12, 1247  
periodic Task (1, 62): 1232, 20, 5, 1252  
periodic Task (3, 41): 1240, 30, 7, 1270  
periodic Task (2, 32): 1252, 40, 12, 1287  
periodic Task (1, 63): 1254, 20, 5, 1272

이 중 Task (1, 60), Task2, Task (1, 63)만 처리되게 됩니다.

Task (1, 60)이 1201의 시간까지 점유를 하고, 비주기적인 Task2가 들어오기 때문에 인터럽트와 비슷한 상황이 발생하고, 따라서 1254의 시간까지 점유를 하게 된다. 따라서 그 사이에 있는 작업들 (빨간 글씨)의 경우 시작 시간이나 주기, 마감 한계의 이유로 수행되지 않는다.

이는 GUI환경에서도 수행되지 않은 작업들이 있다는 것이 확인 가능하다.



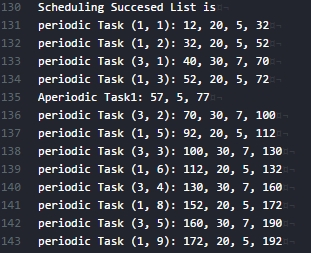
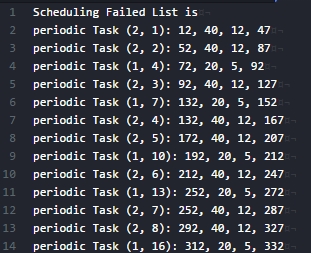
**- USER 알고리즘**

input.txt는 앞선 알고리즘들과 동일하다.

USER 알고리즘을 선택하였다.



스케줄링 버튼을 누른 후, 파일을 확인해보았다.



USER 알고리즘의 경우, 정확히 작업들이 입력되는 시간에만 수행되어야 하기 때문에, 스케줄링 되지 않는 작업들이 많다.

스케줄링 되지 않는 작업들의 이유를 살펴보면 다음과 같다. (다음 장에 이어서)

좌측의 빨간 글씨의 작업들이 수행되지 않은 작업들이고, 우측의 검정 글씨의 작업들이 수행된 작업들이다.

같은 줄에 있는 두 작업(좌측과 우측)은 모두 경쟁관계였고, 우측이 항상 승리하여 수행되었다.

periodic Task (2, 1): 12, 40, 12, 47 periodic Task (1, 1): 12, 20, 5, 32  
periodic Task (2, 2): 52, 40, 12, 87 periodic Task (1, 3): 52, 20, 5, 72  
periodic Task (1, 4): 72, 20, 5, 92 periodic Task (3, 2): 70, 30, 7, 100  
periodic Task (2, 3): 92, 40, 12, 127 periodic Task (1, 5): 92, 20, 5, 112  
periodic Task (1, 7): 132, 20, 5, 152 periodic Task (3, 4): 130, 30, 7, 160

우측과 좌측을 비교해보면, 좌측의 시작 시간에 항상 우측이 점유하거나, 동시에 점유를 시도하고 있고, 점유를 시도하고 있다면 마감 한계가 더 이른 우측이 항상 승리하여 스케줄링이 수행된다.

이는 GUI환경에서도 수행되지 않은 작업들이 있다는 것이 확인 가능하다.



스케줄링에 실패한 작업들이 많음을 알 수 있다.