Test Document

for

<SE Project>

학번 : 2011004028

전공 : 컴퓨터

이름 : 이성훈

교수 : 유민수 교수님

목차 구성

Table of Contents ii

Revision History ii

1. 소개 3

1.1 개략적인 프로그램 설명 3

1.2 요구사항 3

1.3 Design 3

1.4 Scheduling Policy 설명 4

2. 코드 분석 5

2.1 Input 6

2.2 Task generator 7

2.3 Scheduler 8

2.4 Scheduling Policy 9

2.5 output 10

3. 테스트 11

3.1 EDF 11

3.2 RM 12

3.3 User Priority 13

4. Extra 14

4.1 Constraints 14

4.2 느낀점 14

4.3 추가 개선 사항 15

1. 소개
2. 개략적인 프로그램 설명

주어진 입력파일의 정보들을 이용하여 태스크들을 생성하고, 주기, 또는 비주기적인 tasks를 user가 선택한 mode (RM, EDF, User Priority)를 기준으로 스케쥴링을 수행한다. Deadline을 넘은 tasks를 찾아서 어떤 tasks가 missed되었는지 출력하고 사용자가 원하는 구간에서의 tasks scheduling을 확인 할 수 있다. 또한 각 task 별 평균, 최악 응답시간을 알 수 있다.

1. 요구사항

**기능적 요구사항**

-CPU 스케쥴링 시뮬레이터는 데드라인을 어기지 않고 모든 태스크가 스케쥴링이 가능한지 판단해야 한다.

-input file을 읽어 들여 주기적, 비주기적 태스크를 생성할 수 있어야 한다.

-simulator에서 scheduling이 어떻게 전개 되는지 모든 과정 또는 원하는 부분 과정을 출력할 수 있어야 한다.

**비 기능적 요구**

-프로그램이 안정적이어야 하며 오류를 최대한 줄여야 한다.

-시뮬레이팅 시간이 실제 시간과 같이 걸려서는 안되며 실행 후 결과를 바로 확인 할 수 있어야 한다.

3) Design

Simulating

scheduling

output

Deadline check

Task generate

Input tasks arrival

Scheduling quit

위의 activity diagram에서 보이듯이 프로그램의 흐름과 설계는 Input을 받아서 task를 발생시키고 만들어진 tasks을 구조체에 담아서 그 값들을 각각 유지하고 그 값들을 가지고 사용자가 원하는 scheduling policy에 따라 scheduling을 한다. 이 때 Deadline을 check하며 missed된다면 해당 tasks은 놓친 것으로 표시하고 계속 scheduling을 진행해 나간다. 후에 scheduling 결과를 display한다.

1. Scheduling Policy (Wikipedia 참조)
2. EDF

**최단 마감 우선 스케줄링**(Earliest Deadline First Scheduling, 줄여서 EDF 스케줄링)은 [실시간 운영 체제](https://ko.wikipedia.org/wiki/%EC%8B%A4%EC%8B%9C%EA%B0%84_%EC%9A%B4%EC%98%81_%EC%B2%B4%EC%A0%9C)에서 사용되는 동적 [CPU 스케줄링](https://ko.wikipedia.org/wiki/CPU_%EC%8A%A4%EC%BC%80%EC%A4%84%EB%A7%81) [알고리즘](https://ko.wikipedia.org/wiki/%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)의 하나이다. [프로세스](https://ko.wikipedia.org/wiki/%ED%94%84%EB%A1%9C%EC%84%B8%EC%8A%A4)를 [우선순위](https://ko.wikipedia.org/w/index.php?title=%EC%9A%B0%EC%84%A0%EC%88%9C%EC%9C%84&action=edit&redlink=1) [큐](https://ko.wikipedia.org/wiki/%ED%81%90_(%EC%9E%90%EB%A3%8C_%EA%B5%AC%EC%A1%B0))를 통해 수행한다. 스케줄링 이벤트가 일어날 때마다, 큐에서 마감시간이 가장 가까운 프로세스를 탐색하여 다음에 수행되도록 한다. 주기적인 작업 뿐만 아니라 단일 처리기 환경에서 선점형 프로세스들을 스케줄링할 수 있다. 작업(task)이 N개일 때 복잡도는 O(n^2)이다.

1. RM

**비율 단조 스케줄링**(rate-monotonic scheduling, 줄여서 RMS)은 [1973년](https://ko.wikipedia.org/wiki/1973%EB%85%84) [리우](https://ko.wikipedia.org/wiki/%EB%A6%AC%EC%9A%B0)(Liu)와 [래일랜드](https://ko.wikipedia.org/w/index.php?title=%EB%9E%98%EC%9D%BC%EB%9E%9C%EB%93%9C&action=edit&redlink=1)(Layland)가 제안한 [실시간 시스템](https://ko.wikipedia.org/wiki/%EC%8B%A4%EC%8B%9C%EA%B0%84_%EC%8B%9C%EC%8A%A4%ED%85%9C)을 위한 스케줄링 정책이다. **비율 단조 기법**이라고도 한다.[[1]](https://ko.wikipedia.org/wiki/%EB%B9%84%EC%9C%A8_%EB%8B%A8%EC%A1%B0_%EC%8A%A4%EC%BC%80%EC%A4%84%EB%A7%81#cite_note-1) **비율 단조 분석**(rate-monotonic analysis, 줄여서 RMA)은 RMS의 배경이 되는 이론으로서, 시간 당 CPU 사용률을 계산하여 [프로세스](https://ko.wikipedia.org/wiki/%ED%94%84%EB%A1%9C%EC%84%B8%EC%8A%A4)들을 이상없이 수행할 수 있는지를 알아보는 일을 말한다.

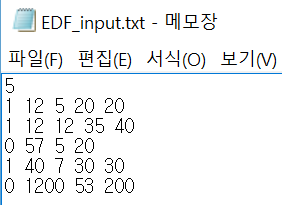
수행 주기가 가장 짧은 프로세스에 가장 높은 [우선순위](https://ko.wikipedia.org/w/index.php?title=%EC%9A%B0%EC%84%A0%EC%88%9C%EC%9C%84&action=edit&redlink=1)를 부여하는 방식이다. 따라서, 비율(rate, 단위시간당 프로세스 수행 횟수, 수행 주기의 역수)과 [우선순위](https://ko.wikipedia.org/w/index.php?title=%EC%9A%B0%EC%84%A0%EC%88%9C%EC%9C%84&action=edit&redlink=1)의 관계를 그래프로 나타내면 우상향 직선형이 된다. 이렇게 비율에 따라 우선순위가 단조롭게 증가하는 추세를 보인다고 하여 '비율 단조'라는 이름이 붙게 되었다. 프로세스에 부여하는 우선순위의 변동이 없기 때문에, [정적 스케줄링 정책](https://ko.wikipedia.org/wiki/%EC%A0%95%EC%A0%81_%EC%8A%A4%EC%BC%80%EC%A4%84%EB%A7%81_%EC%A0%95%EC%B1%85)이라고 할 수 있다. 그렇지만 상당히 효율적이라서 아직도 널리 사용되고 있다. RMS를 사용하는 운영체제는 일반적으로 [선점형](https://ko.wikipedia.org/wiki/%EC%84%A0%EC%A0%90%ED%98%95)이고, 응답시간에 대해 결정적(deterministic)인 특징이 있다.

1. User Priority

EDF에서는 짧은 deadline, RM에서는 짧은 주기를 기준으로 높은 우선순위를 부여했다면 User Priority는 애초에 User가 Priority를 지정을 해줘서 가장 우선순위가 높은 프로세스를 먼저 수행하겠다는 알고리즘이다.

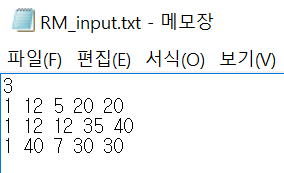
1. Code Analysis
2. Input

EDF\_input.txt



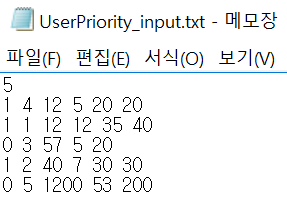
첫 번째 줄은 tasks의 개수를 받고 각 줄마다 주기적 task 인지 비주기적 task 인지(주기 = 1 , 비주기 = 0), Arrival time, execution time, Deadline, Period를 나타낸다. 비주기 함수는 period를 쓰지 않는다.

RM\_input.txt



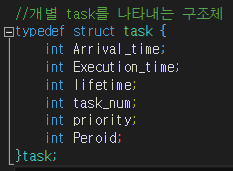
RM도 기본 원칙은 EDF의 input file과 일치한다. 그러나 RM은 주기 task만을 고려한다.

UserPriority\_input.txt

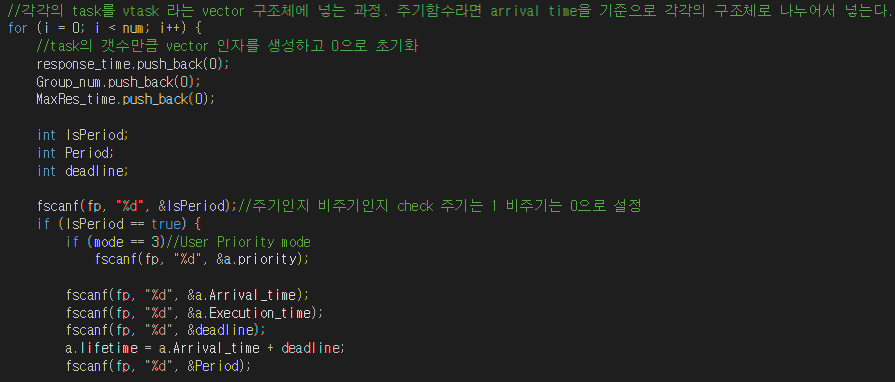


User Priority 는 위의 두 input file과는 다르게 두 번째 항으로 우선순위를 가진다. 이 때 우선순위는 simulator의 input을 넣는 이용자가 임의로 넣어줄 수 있다.

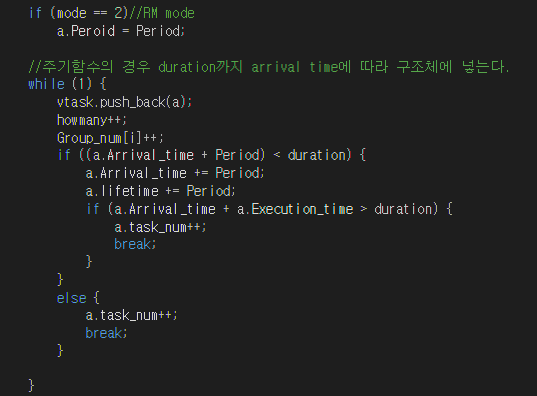
1. Task generator



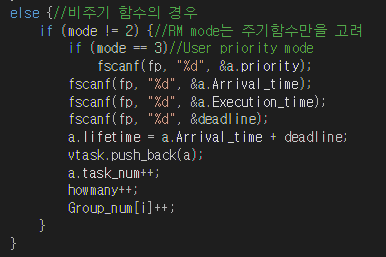
Tasks 구조체는 arrival\_time, execution\_time, lifetime ( arrival\_time + deadline), task\_num, priority, period 로 구성된다. lifetime 값과 arrival time 값을 기준으로 task가 현재 놓치지 않았는지를 판단한다. 각각의 mode로 sorting이 되는데 EDF는 lifetime을 기준으로, RM은 period를 기준으로, User Priority는 priority값을 기준으로 sorting을 하고 simulating을 한다.



Vector <구조체> 형식의 vtask를 이용하여 각각의 task들을 벡터에 넣는다.

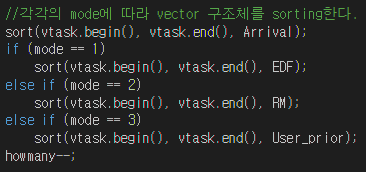


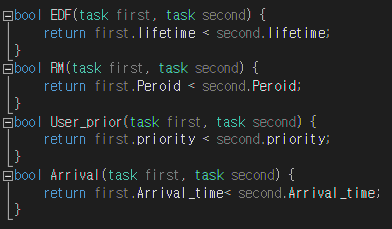
Task를 생성하는 부분에서 주기적인 task를 각각의 비주기적 tasks로 나누는 역할을 한다. Arrival\_time을 duration보다 작은 모든 주기에서 고려를 함으로써 비주기적인 tasks을 vtask라는 구조체 벡터에 차례로 넣는다. 이 때 고려해야 하는 것은 duration을 넘지 않는 것인데 도착시간 + 실행시간이 구간을 넘지 않아야 한다.



비주기 tasks의 경우에는 바로 구조체 벡터에 넣을 수 있다.

1. Scheduling policy

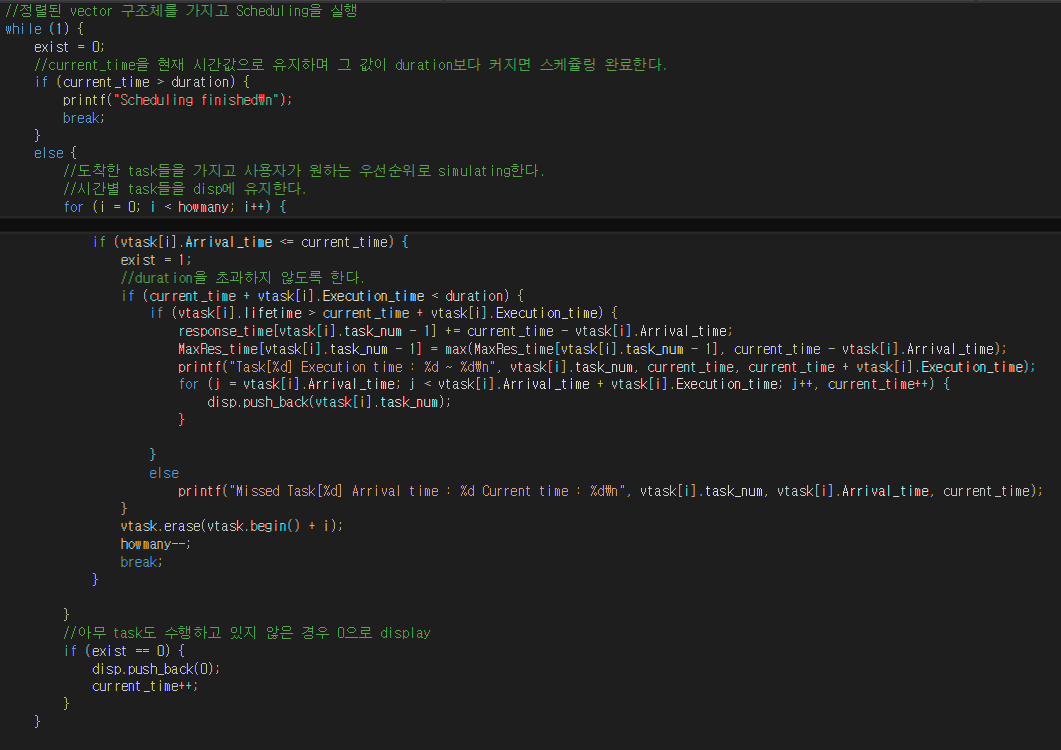




Algorithm 헤더파일에 있는 sort 함수를 이용하여서 구조체의 arrival\_time을 기준으로 sorting 한번 하고 사용자가 정한 mode에 따라 우선순위를 기준으로 sorting을 한다. 원하는 Scheduling policy에 따라 sorting을 했으면 순차적으로 탐색하며 simulating이 가능 해진다.

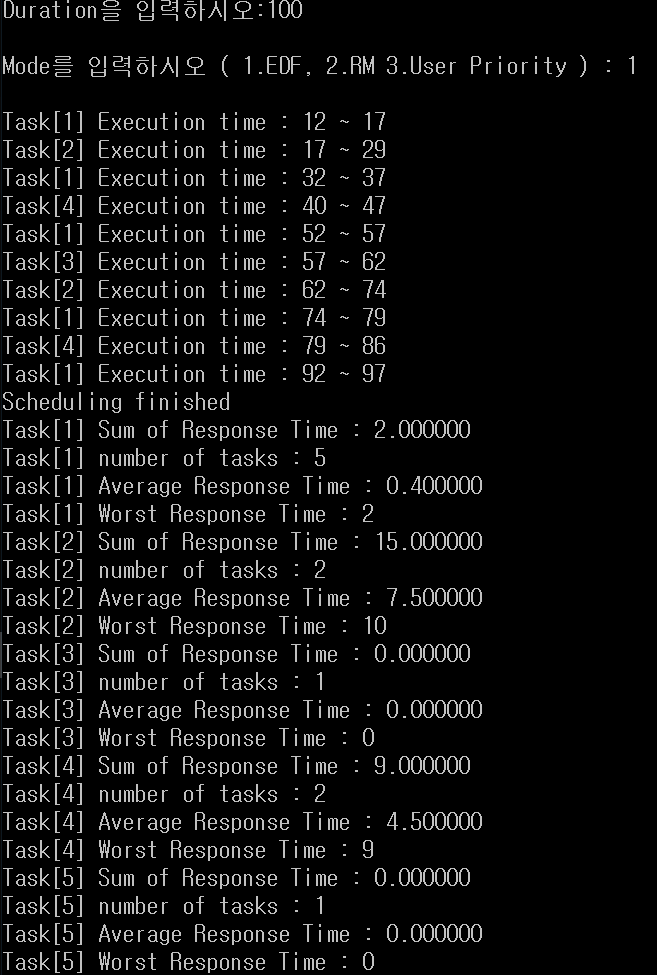
1. Scheduler

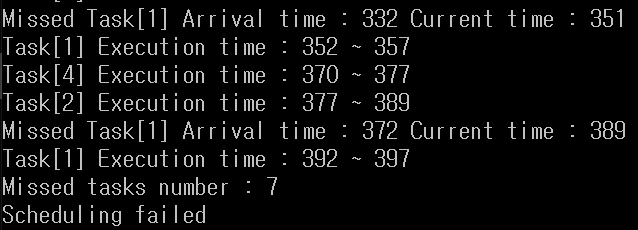
Scheduler에서 가장 중요하게 고려해야 할 점은 current\_time이라는 변수의 증가이다. Current\_time이라는 변수를 이용하여서 현재 시간을 유지해서 그 해당시간에 어떤 task가 가장 사용자가 정한 우선순위에 적합한지를 따진 후 해당하는 task를 수행한다. 초기 duration을 정했다면 그 duration까지의 모든 task을 대상으로 scheduling을 진행한다. 이때 scheduling의 결과가 저장되는 부분은 disp라는 vector이다. Disp는 나중에 스케쥴링 결과를 출력하는데 사용한다. Average response time 과 Worst response time을 유지하는 벡터를 따로 만들어서 저장한다. Simulator를 만들 때 vector를 많이 사용했는데 이는 sorting을 쉽게 할 수 있고 자체적으로 동적인 메모리 할당이 되어지기 때문에 메모리 관리가 쉬워서이다.



1. Output

출력은 표준 출력을 이용한다. Cmd 창에 duration과 mode를 입력하면 파일을 입력 받아서 Simulating한 결과를 표준 출력한다. 이때 각각의 task가 어느 시간대에 실행이 되었는지를 시각적으로 알 수 있게 하였고 missed된 task가 있다면 그 task가 몇번째 task인지, 어느 시점에서 missed 되었는지, 언제 도착한 task인지를 표시해 준다. Scheduling이 완료되었다면 Scheduling finished를 출력한다. 만약 missed된 task가 5개가 넘어가면 스케쥴링 실패로 정하고, missed task의 갯수와, scheduling failed를 출력한다. 마지막으로 사용자가 원하는 구간을 입력하면 해당 구간에 어떤 task가 실행되었는지를 시각적으로 표현한다.



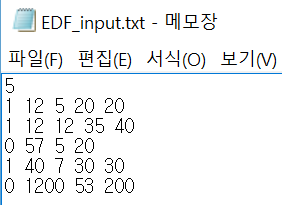




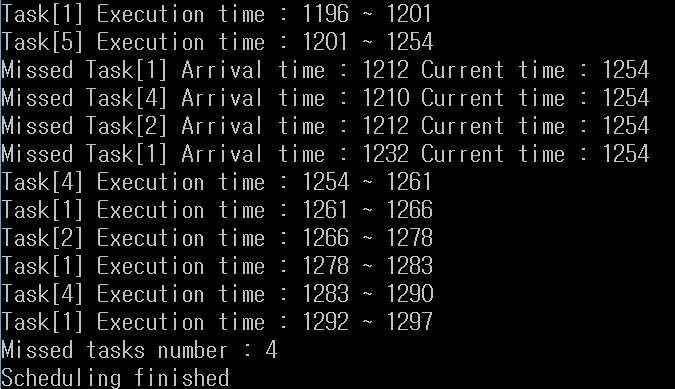
1. TEST
2. EDF

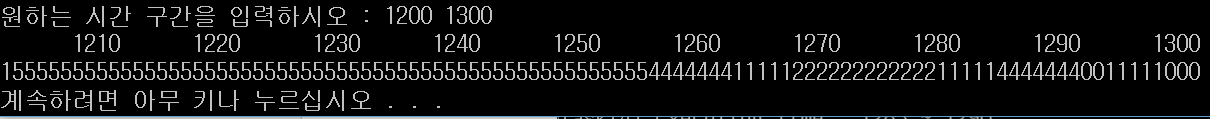
Duration : 1300으로 실행

Input>



Task[5] 가 실행되는 구간에서 task 1, 4, 2, 1 이 missed 된 것을 알 수 있음. 이유를 자세히 알아보기 위해서 원하는 구간의 task display를 해 보았음.





1200 ~ 1300에 도착하는 tasks에는

1번 task 1212, 1232, 1252, 1272, 1292

2번 task 1212, 1252, 1292

3번 task x

4번 task 1210, 1240, 1270

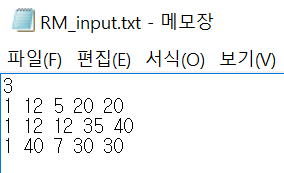
5번 task 1200

까지 있다. EDF는 deadline이 가장 짧은 것들을 먼저 실행하는데 current time이 1201일 때에는 5번을 제외하고 아무것도 도착하지 않았으므로 5번 task가 수행된다. 5번 task가 1254까지 수행이 되는데 이 때 도착한 task들은 우선순위가 높아도 실행이 되지 않는다. 따라서 1번 tasks 1212, 1232, 2번 task 1212, 4번 task 1210, 1240이 이 구간에 존재하는데 여기서 4번 task 1240만이 5번 task가 끝나고도 실행해도 deadline이 초과하지 않으므로 5번이 끝나자마자 4번 task가 수행되게 된다. 4번의 1240을 제외한 나머지 task들은 deadline이 초과되므로 모두 missed되게 된다.

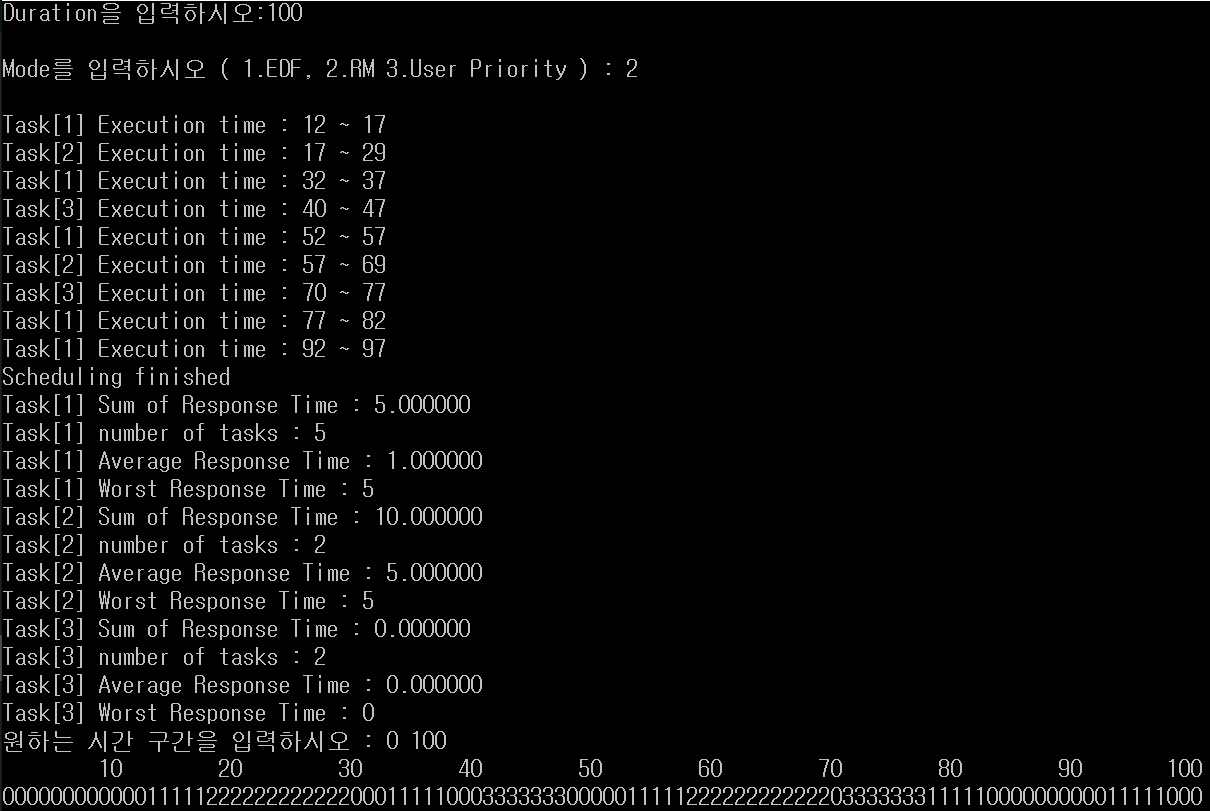
1. RM

Duration 100으로 실행

Input>



RM방식은 현재 시간에 도착한 task들 중에서 주기가 가장 짧은 것을 먼저 실행한다.

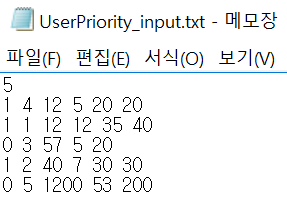


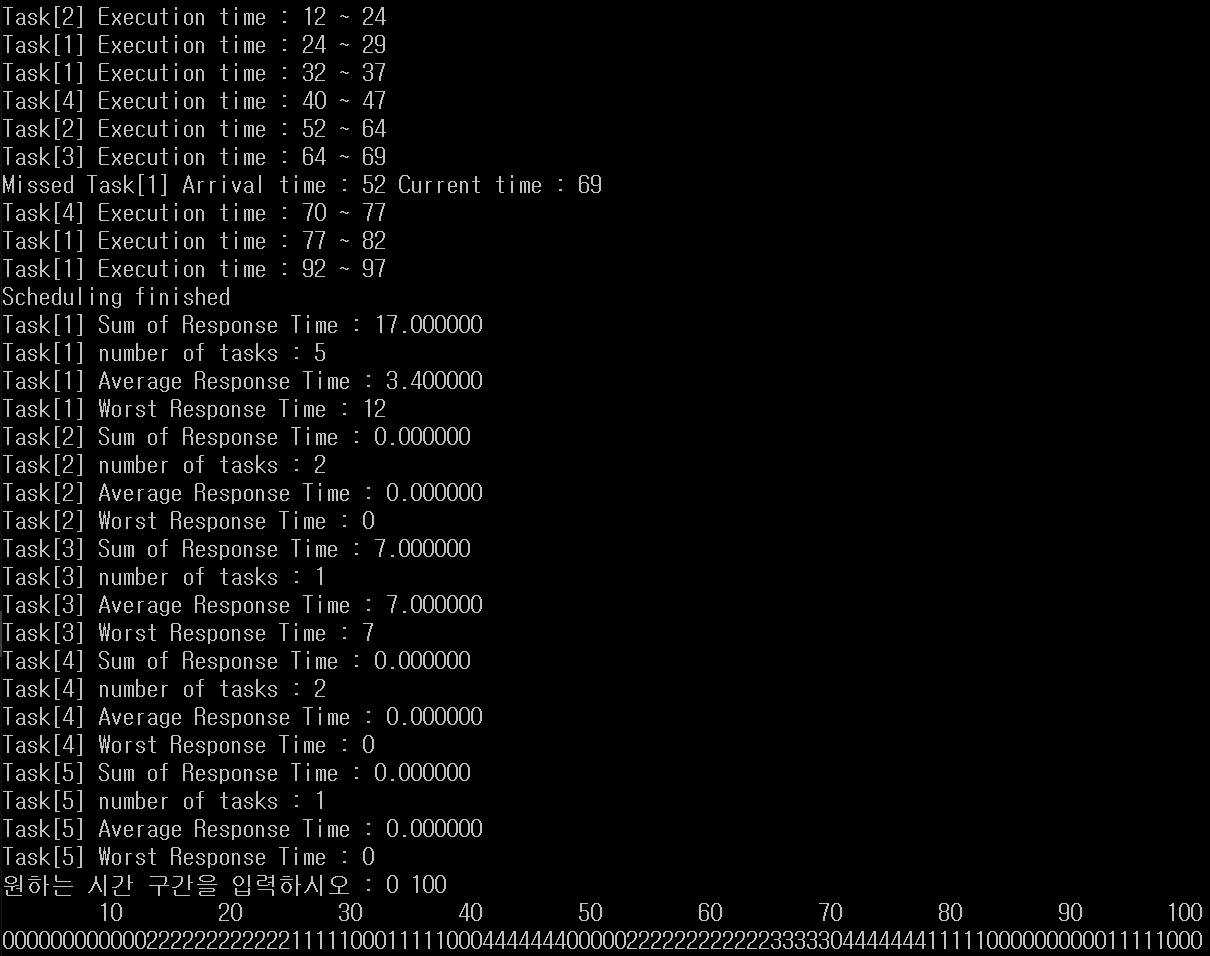
처음에 1번 task과 2번 task 둘 다 12초에 도착하게 되는데 이때 1번 task의 주기가 20 이고 2번 task의 주기가 40 이기 때문에 1번 task가 실행되게 된다. ***여기서 문제점은 1번 task와 2번 task의 도착시간이 같고 1번 task의 주기와 2번 task의 주기가 40이기 때문에 매 주기마다 (12, 52, 92 …) 1번과 2번 task는 경쟁을 하게 된다.*** Response time의 결과를 봤을 때 2번 task는 매번 1번 task의 실행시간인 5초를 wait하고 시작하는 것을 알 수 있다. 이 문제는 실무상에서 봤을 때 waiting time이 커서 굉장히 비효율적이기 때문에 추가적으로 개선을 할 필요가 있다.

1. User Priority

Duration 100으로 실행

Input>





User Priority 에서는 2번째 col을 priority 값으로 갖는다.

User priority simulating에서 중요하게 생각할 점은 priority 값을 지정하는 부분이다. User가 임의로 priority의 결정권을 갖게 하는 데에 자율성을 두지만 이는 똑똑한 user를 가정한다. 단순히 중요한 task를 우선순위에 두게 되면 task 처리의 비효율성이 매우 높아지게 될 가능성이 크고 실제 simulating에서는 risk가 커질 우려가 있다.

1. Extra
2. Constraints

Simulator를 작성하며 가장 우려사항이 큰 것은 이상적인 상황을 가정하며 설계를 하고 프로그램을 작성한다는 것이다. 하지만 실제 simulating 또는 실제로 scheduling을 진행해 나가는 과정에서는 여러 상황에서 delay가 발생할 수도 있고 예상치 못한 오류가 발생할 수 있는데 이를 고려하며 신뢰성 있는 설계를 해 나가는 것에서 어려움을 느낀다.

1. Requirements를 받고 design을 하고 소프트웨어 개발을 하고 test를 하는 과정에서 실제 requirements와 design, 그리고 개발을 하는데 그것이 완벽하게 부합하기가 어렵다는 생각이 들었다. 요구사항에 따른 제한적인 기간 안에 개발을 해야 하는데 시작하기 전에 고려했던 것이 막상 개발에 들어가니 생각보다 어렵고 불가능한 경우가 생기는데 이것이 어려움이 있었다.

또한 개발을 하며 추가하고 싶은 사항이 생기고 부족한 점이 보이면 그것을 개선해나가야 하는데 이것을 하는 시간적인 부담이 너무 크고 테스트의 비용( 시간, 한정적인 실험군 등등)이 많이 든다.

1. 추가 개선 사항

먼저는 좀 더 세련된 그래픽으로 표현하면 좋을 것 같다는 생각이 들었다. Graph나 표를 활용해서 그것을 output으로 제공하면 훨씬 시각적으로 좋을 것 같고 user에게 도움이 될 것 같다는 생각이 들었다.

두 번째로는 RM방식에서 계속되는 충돌이 지속될 경우 주기는 유지하되 도착시간을 다르게 하는 방법을 강구하면 좋을 것 같다는 생각을 했다. 계속되는 비효율적인 충돌을 막는 것이 RM방식의 꼭 필요한 개선 방향이라고 생각한다.

또한 Scheduling의 효율성을 check하는 점을 추가하면 좋을 것 같다. Response time으로 물론 check를 하고 있지만 현재 scheduling에서 CPU가 놀고 있는 시간 (본 프로그램에서는 0으로 표현하였다.)이 얼마나 되는지를 check해서 CPU 효율성이 얼마나 되는지를 check하고 어떤 scheduling이 유리한지를 알려줄 수 있는 기능을 추가한다면 좋을 것 같다.