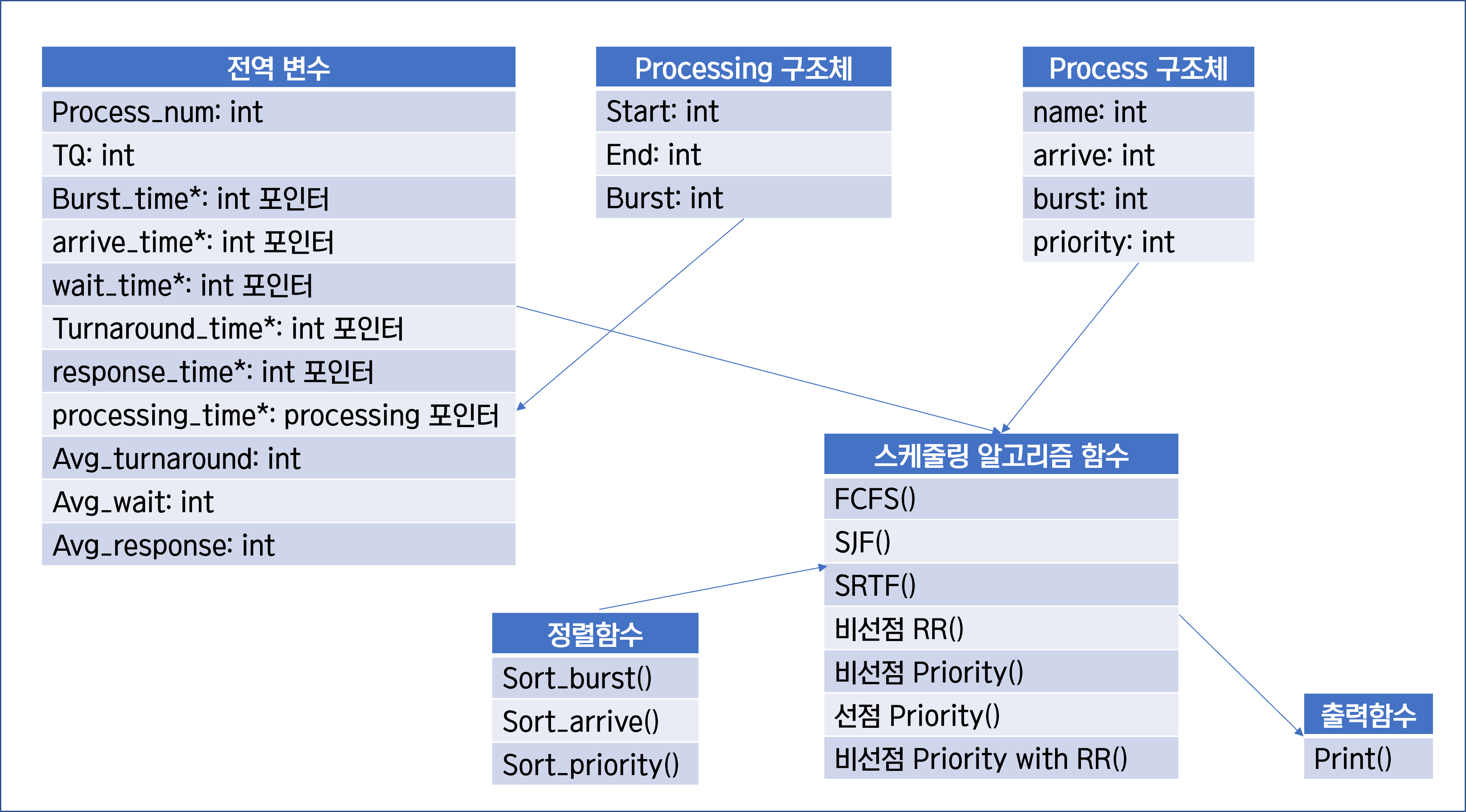
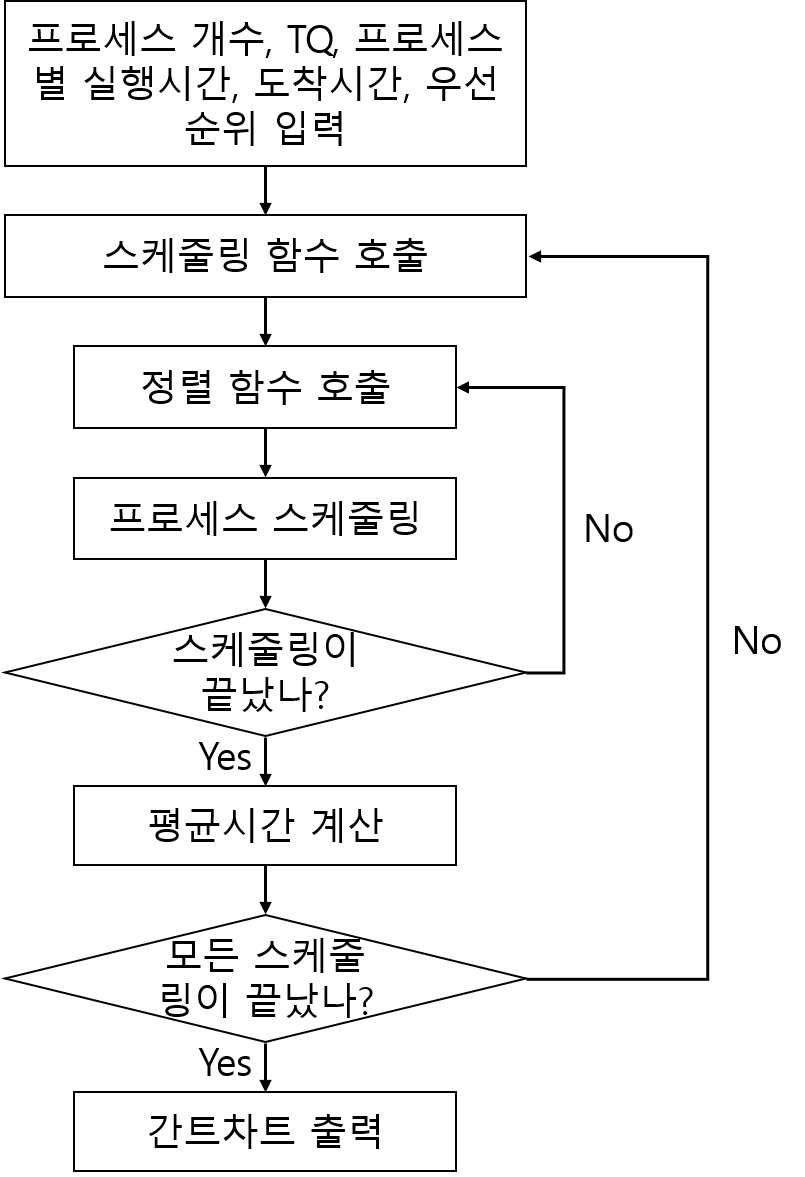
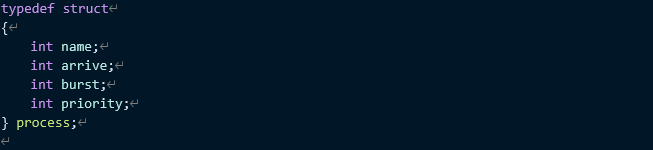
**운영체제 팀프로젝트 5팀**

* 최종 보고서−

박혜연(팀장)  
이지수  
김현채

**목차**

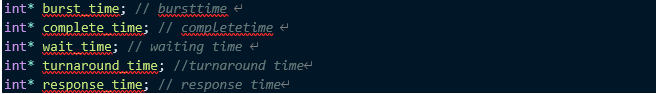
1. 팀 GitHub
2. UI 실행 URL
3. 팀원 역할
4. 개발언어
5. 함수/클래스 구성도/다이어그램
6. 자료구조/주요변수 설명
7. 각 알고리즘 핵심부분 코드 설명
8. 개발한 UI 특징
9. 각 알고리즘 실행 결과 캡쳐 및 설명
10. 팀원별 느낀 점

* **팀 GitHub:** <https://github.com/gpdus0321/OPERATING-SYSTEMS_5team.git>
* **UI 실행 URL:** <http://r4bb1t.pythonanywhere.com/>
* **팀원 역할**   
  이지수 – FCFS, SJF, SRTF 개발  
  박혜연 – RR, 비선점 Priority 개발  
  김현채 – UI 개발
* **개발언어:** C++ -> 최종적으로 python으로 변경
* **함수/클래스 구성도/다이어그램:**
* **자료구조/주요변수 설명:**
* 자료구조  
  
* 프로세스 구조체  
  

프로세스 별 이름, 도착 시간, 버스트 시간, 우선 순위 데이터를 저장하는 구조체입니다.

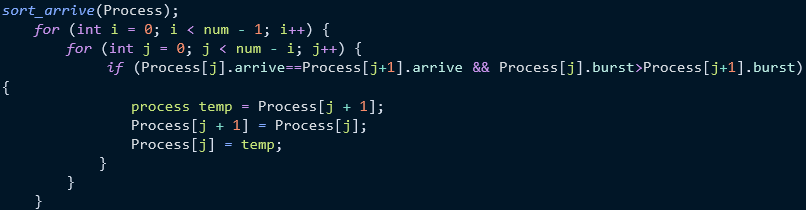
* 프로세싱 구조체  
  

간트 차트에 들어갈 시간 별 실행되는 프로세스를 표현하는 구조체입니다. 시작 시간, 끝 시간, 버스트 시간이 들어 있습니다.

* 각 프로세스 별 실행 시간 배열  
  

프로세스 별로 실행되며 걸린 wait time, turnaround time을 담는 배열입니다. 이 값을 모두 더하고 프로세스 개수로 나눠서 평균 시간을 구합니다.

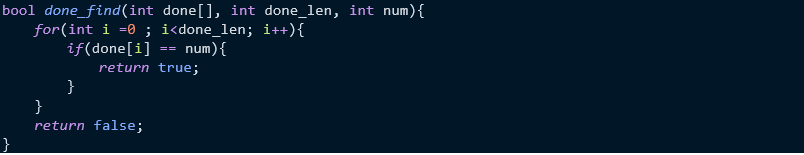
* **각 알고리즘 핵심부분 코드 설명:**
* **FCFS**FCFS알고리즘의 경우 프로세스 구조체를 모두 도착 시간 순서 대로 정렬 함수로 정렬합니다. 선점 기법을 사용하지 않기 때문에, for 문을 사용해서 도착 순서대로 프로세스에 접근해서 한 번만 시간을 계산합니다
* **SJF**

〈프로세스 정렬 부분〉

프로세스들을 도착순서대로 정렬하고, 도착 시간이 같으면 더 짧은 것을 골라 가장 먼저 실행할 프로세스를 고르도록 했습니다.

  
〈priority queue 사용 부분〉

FCFS와는 달리 도착 시간이 빨라도 프로세스이 길이가 짧은 것이 사용되기 때문에, priority queue를 사용해서 현재 시간보다 작은 것들을 priority queue에 넣고 burst time이 작은 것을 골라서 사용했습니다.

  
〈중복 검사 함수〉

이미 사용한 것들을 중복해서 priority queue에 추가하는 일을 막기 위해서 이미 사용한 프로세스의 number를 저장하는 int 배열을 선언해서 priority에 추가하기 위한 조건으로 추가했습니다.

* **SRTF**

SJF와 처음 실행할 프로세스를 선정하기 위한 과정까지는 동일합니다.

  
〈while문 조건과 변경 부분〉

다만 SJF는 선점이 없어서, 프로세스 별로 실행을 했다면, SRTF는 FCFS를 통해서 종료시간을 변수에 저장하고, while 문으로 시간을 1씩 증가시켜가며 종료시간에 되면 스케줄링을 종료하는 방식을 사용합니다.

  
〈tmp(priority queue의 top)과 현재 프로세스를 비교하는 부분〉

  
〈현재 프로세스의 남은 실행시간을 비교하는 부분〉

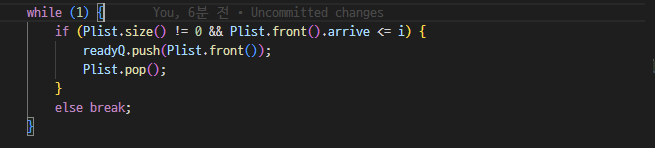
실행 중이던 프로세스가 다른 것으로 교체되는 경우는 현재 priority에 들어온 다른 프로세스의 burst time이 더 짧은 경우와 현재 실행 중이던 프로세스의 burst time이 0이되는 경우 2가지가 존재합니다. While 문에서 2가지 경우를 if문으로 짜서 프로세스의 선점을 반영했습니다.

  
〈실행 시 초기 버스트 값을 저장〉

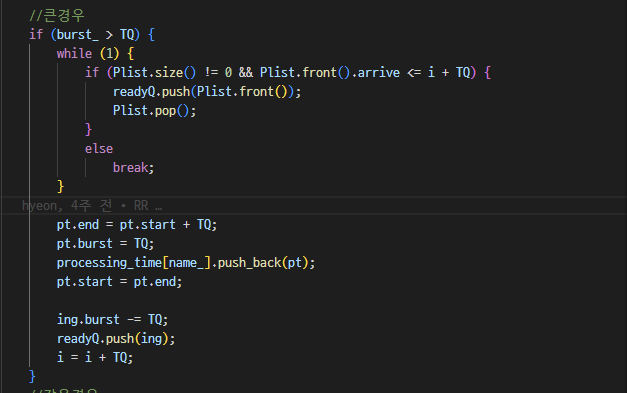
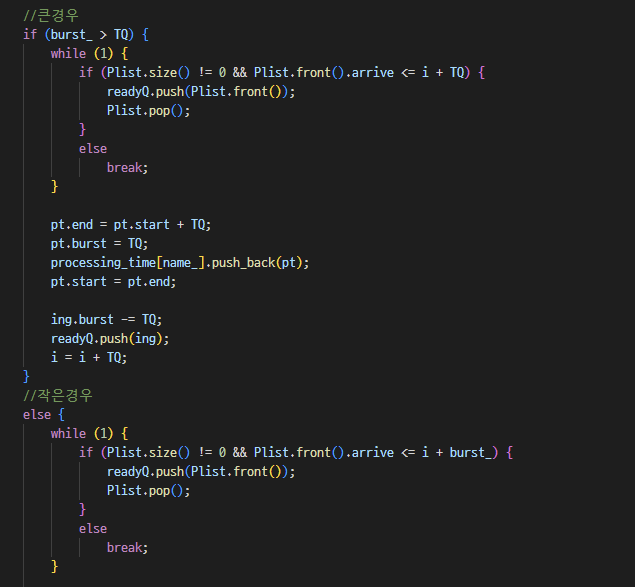
  
〈종료 시 저장한 버스트 값을 복구〉

SRTF의 경우 실행 과정에서 프로세스의 burst 값이 줄게 되는데, 같은 프로세스 배열을 다른 함수에서도 사용함으로 종료 시에 원래의 burst 값으로 다시 채워줄 필요가 있었습니다.

* **RR**RR은 도착시간 순서에 따르면서 TQ시간마다 실행 프로세스가 변경되는 것입니다. 그래서 먼저 sort\_arrive()함수를 이용하여 도착시간 순서대로 프로세스들을 정렬해줍니다.

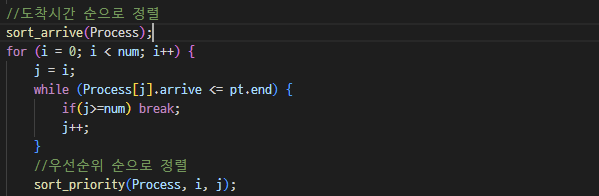
  
〈queue에 담기〉

그리고 진행중인 시간전에 도착한 프로세스들을 TQ시간마다 바뀌면서 진행되어야 하기 때문에 진행시간전에 도착한 프로세스들을 queue에 넣어 줍니다.

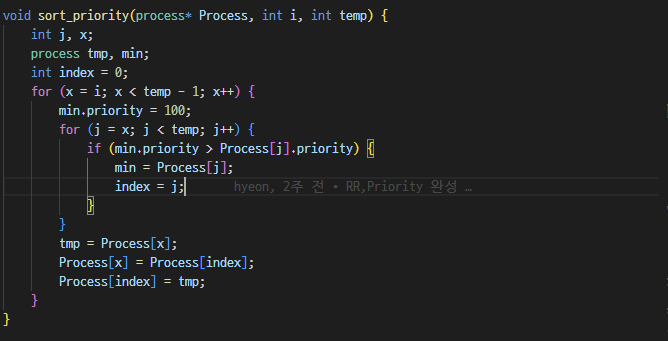
  
〈진행중인 프로세스의 burst가 TQ보다 큰 경우〉  
  
  
〈진행중인 프로세스의 burst가 TQ보다 작 경우〉

그 뒤에 실행될 프로세스의 burst시간이 TQ보다 작은 경우와 큰 경우 두가지로 나눠줍니다. 왜냐하면 TQ보다 큰 경우는 queue에서 pop해주고 burst시간을 TQ만큼 빼주고 다시 queue에 넣어주고 진행하기 때문입니다. 작은 경우에는 그 프로세스는 실행이 끝난 것 이므로 다시 queue에 넣어줄 필요가 없으며 그 프로세스의 응답, 대기, 반환 시간을 계산해주면 됩니다.

* **비선점 Priority**

  
〈priority 정렬 방법〉

비선점 priority는 우선순위 순서대로 진행되는 것입니다. 그렇기 때문에 먼저 도착 시간 순으로 정렬하고, 진행중인 시간에 대기중인 프로세스가 여러 개라면 거기서 우선순위 순으로 정렬하도록 하였습니다. 그 뒤에는 비선점이기 때문에 실행되는 프로세스 순서대로 진행하면 됩니다.

  
〈sort\_priority 함수〉

sort\_priority() 함수는 process[i]부터 process[temp]까지의 프로세스를 우선순위순서로 정렬하는 함수입니다.

* **개발한 UI 특징:** 각 스케줄러 별 간트 차트, 반응 시간 등을 깔끔하게 한 눈에 볼 수 있도록 나눠서 보이도록 하였습니다. 또한, 파이썬 플라스크를 이용하여 웹으로 구현하였고, 배포까지 하여, URL만 있다면 어떤 환경에서도 간편하게 사용할 수 있습니다. 그렇기 때문에 인터넷만 사용 가능하다면 컴퓨터뿐만 아니라 스마트폰이나 테블릿으로도 사용이 가능하여 접근성이 좋습니다. 입력 및 실행의 과정이 간결하고 빠르게 확인 가능합니다.
* **각 알고리즘 실행 결과 캡쳐 및 설명:**







시간할당량과 원하는 만큼의 프로세스를 추가해주고 계산하기를 누르면 각 알고리즘의 간트 차트가 보여집니다. 프로세스 그래프를 그려주고 진행되는 프로세스의 ID를 보여줍니다. 그리고 프로세스별 대기, 반환, 응답 시간을 보여주고, 평균 시간들을 보여줍니다.

* **팀원별 느낀 점**

**박혜연-** 이론으로 이해를 다 하였다고 생각하였는데, 실제로 코드를 작성하다 보니 헷갈리는 부분도 있었고, 머리로 이해한 것을 코드로 작성하는게 생각보다 어려웠습니다. 스케줄링 알고리즘을 구현하면서 정렬에 대해서도 더 잘 이해하게 되었고, 어떤 상황에서 어떤 정렬을 사용하면 좋을지도 다시 생각해보게 되었습니다.  
그리고 팀으로 무언가를 구현해본 것은 이번이 두번째인데 이번에 다이어그램을 처음으로 그려보았고, 함께 개발을 할 때는 다이어그램이 있어야 서로의 코드를 이해하기 쉽고, 통일성이 있으며, 다음 사람이 이어서 개발하기 쉽다는 것을 알게 되었습니다. 또한, 소통을 계속하여, 중간중간 진행상황보고를 해주어야 일정에 맞춰서 끝낼 수 있고, 원활하게 진행된다는 것을 다시 한번 깨달았습니다. 그런 점에서 이번 팀프로젝트는 일정에 맞춰서 원활하게 진행된 것 같고 큰 이슈 없이 잘 진행되어 뿌듯하였습니다.

**이지수-** 스케줄링 알고리즘을 짜면서 어려워 보이지 않는 기능임에도 고려할 사항이 많음을 느꼈습니다.

**김현채-** 원래 JS로 웹을 구현하기 위해 C++코드를 웹 어셈블리로 컴파일하려고 하였으나 쉽지 않아 파이썬 플라스크로 웹을 구현했습니다. 처음에는 C++ 코드를 공유 라이브러리로 만들어 사용하려고 노력했으나 쉽지 않아, 처음 써보는 파이썬으로 코드를 변환해 사용했습니다. 다른 분들이 쓴 코드를 이해하기 위해 노력했고, 파이썬의 문법과 원리를 익힐 수 있었습니다.