
== REPORT ==

< OS Term Project >

과 목		운영체제			
담당 교수		이화민			
제출 일자		2022.06.17			
팀		8 조			
팀 원	학과	산업경영공 학부	디지털경영 학과	바이오의공 학부	컴퓨터학과
	학번	2020170812	2019390709	2019250074	2020320051
	이름	임정섭(조장)	박상호	이영섭	진시윤

Github URL: <https://github.com/jseop-lim/cpu-scheduling-simulator-python>

- INDEX -

I.	개발 과정	3
i.	팀원 역할	3
ii.	팀원 Github.....	3
iii.	협업도구	8
II.	프로그램 소개	13
i.	개발언어.....	13
ii.	개발일정.....	14
III.	프로그램 구조	14
i.	프로그램 설계	14
ii.	클래스 다이어그램	15
iii.	자료구조 / 주요변수	18
iv.	알고리즘 핵심코드	26
v.	UI 특징	35
IV.	개발 결과	36
i.	알고리즘 실행 결과.....	36
V.	느낀 점.....	43

I. 개발 과정

i. 팀원 역할

박상호: 보고서 작성, PPT 작성

이영섭: 보고서 작성, UI 설계, UI 개발, 테스트 케이스

임정섭: 조장, 발표자, 프로세스, 클래스 및 스케줄러 구성, 전반적인 개발 프로세스 관리 및 조율

진시윤: 간트차트 구현, UI 설계, UI 개발, GUI 프로그래밍

ii. 팀원 Github

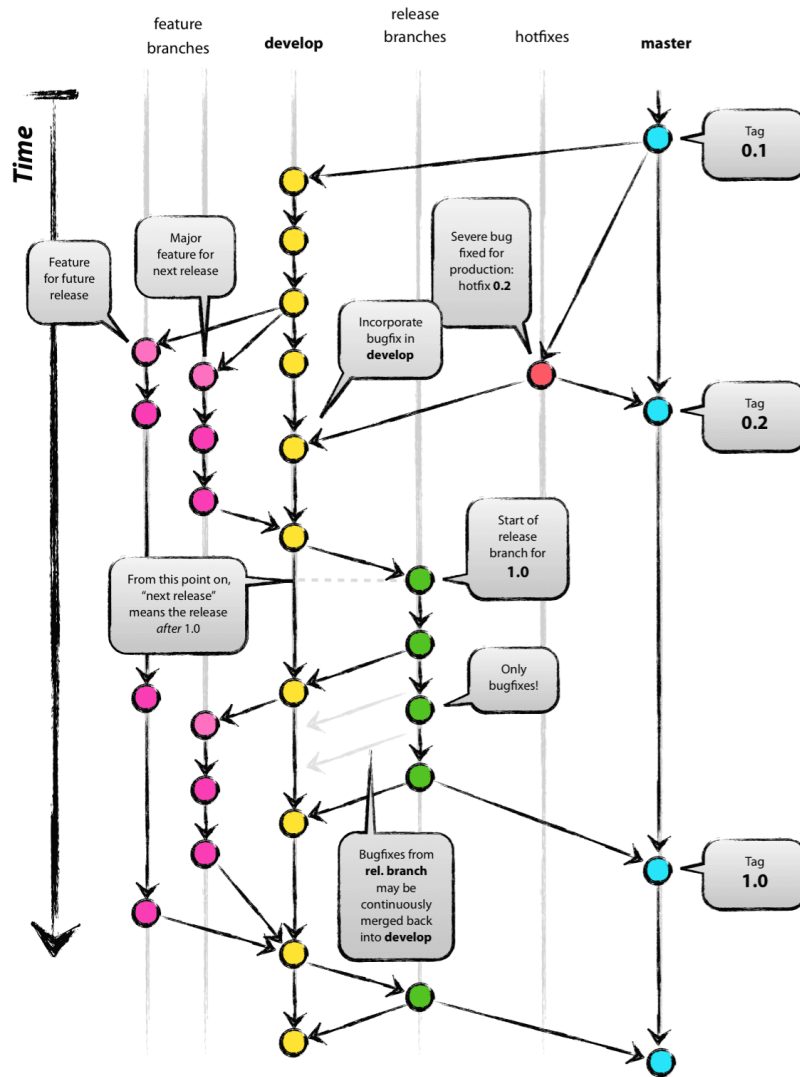
Configuration Management

Branch Convention

Branch Strategy

Branch 관리 전략으로서 Git-Flow의 간략화된 방식을 채택하였다.

Git-Flow에 대해 간략하게 설명하자면 branch를 역할 상 5가지로 분리하여 관리하는 것이다.



5가지 branch로는

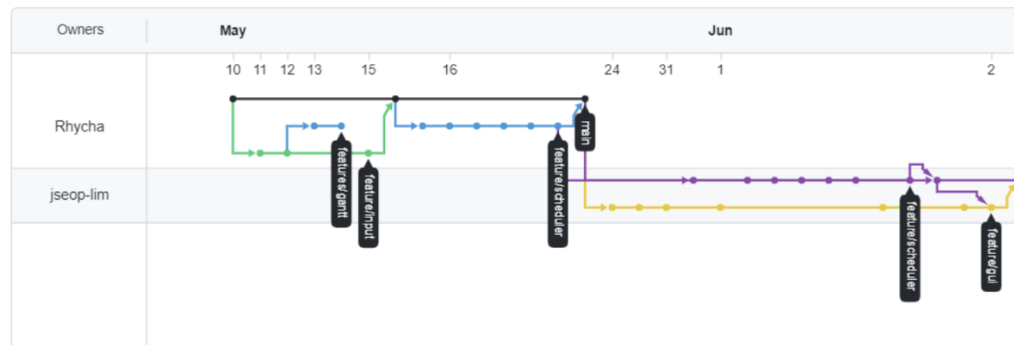
- ① master(출시 버전)
- ② hotfixes(출시 후 문제 발생 시 수정)
- ③ release(출시 이전 준비단계)
- ④ develop(개발이 이루어지는 중심 branch)
- ⑤ feature(각 기능을 개발하는 branch, 개발이 완료되면 develop에 merge 후 삭제됨)

가 있다.

우리는 출시를 목적으로 하는 것이 아니므로 main을 develop branch로 두고, feature 각각을 개발하여 main에 pull request를 통해 merge하였다.

Network graph

Timeline of the most recent commits to this repository and its network ordered by most recently pushed to.



Brach Strategy에 따른 Branch Graph 변화 모습

Branch Naming Rule

유형/내용으로 하였다.

- 유형: feature(기능 추가), fix(오류 수정)
- 내용: 단위는 Notion To-do 칸반 보드의 Task 하나, 즉 Issue 하나로 하였다.



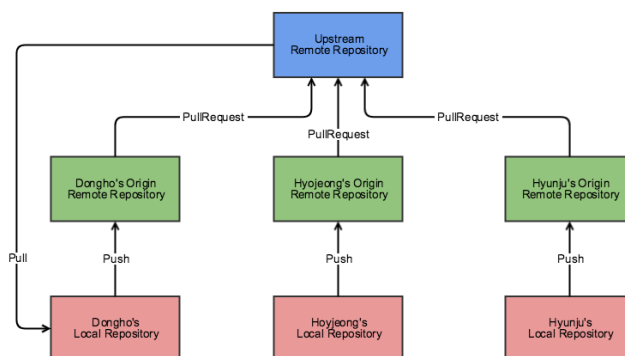
이슈의 예

ex) feature/input, feature/gantt, fix/fcfs

Labels	Milestones	Search all labels
9 labels		
bug	Something isn't working	
documentation	Improvements or additions to documentation	
duplicate	This issue or pull request already exists	
enhancement	New feature or request	
good first issue	Good for newcomers	
help wanted	Extra attention is needed	
invalid	This doesn't seem right	
question	Further information is requested	
wontfix	This will not be worked on	

issue를 구분할 수 있는 label들

Pull Request Convention



출처: 우아한 형제들 기술 블로그

Git Repository 구성

Upstream Remote Repository, Origin Remote Repository, Local Repository 3부분으로 나뉜다.

Upstream Repository는 팀원들이 다같이 공유하는 저장소로 최신 소스코드가 저장되어 있는 원격 저장소이다. Origin Repository는 Upstream Repository를 Fork한 원격 개인 저장소입니다. 마지막으로 Local Repository는 각 개인의 컴퓨터에 저장된 개인 저장소이다.

Pull Request 워크플로우

1. Local Repository에서 작업을 완료한 후 작업 브랜치를 Origin Repository에 push 한다.


- A. 작업 완료의 기준은 한 Issue에 대한 기능 구현 및 수정이 완료되었을 경우를 의미한다

0 Open ✓ 4 Closed		Author	Label	Projects	Milestones	Reviews	Assignee	Sort
<input type="checkbox"/>	Feature/gui	#6 by jinSY515 was merged 8 days ago						
<input type="checkbox"/>	Feature/scheduler	#5 by jseop-lim was merged 8 days ago • Approved						3
<input type="checkbox"/>	Feature/scheduler	#4 by jseop-lim was merged 24 days ago • Approved						3
<input type="checkbox"/>	Feature/input	#3 by jseop-lim was merged 25 days ago						1

Pull Requested Branches 목록

2. Github에서 Origin Repository에 push한 브랜치를 Upstream Repository로 merge하는 Pull Request를 생성하고 코드리뷰를 거친 후 merge 한다.

- A. Reviewer로 팀원 한 명이상을 등록한다.
B. Pull Request를 Merge하는 일은 반드시 해당 Pull Request 담당자가 한다.


jseop-lim commented 9 days ago • edited
Owner

개요

scheduler 클래스 변경 및 SRTF scheduler 완성

작업 사항

- FCFS가 상속하는 Scheduler 추상클래스 정의
- Scheduler class에 avg_(time) property method 추가
- SRTF 완성
- PriorityPreemptiveRR을 PriorityRR로 변경

코드 리뷰 요청 사항

@jinSY515 avg_(time) 코드 확인 부탁드립니다.

@Rhycha 테스트 케이스 입력 부탁드립니다.

코드 리뷰 반영 사항

@sangho00 input2.txt 프로세스 개수 5로 수정

Pull Request를 Merge하는 일은 반드시 해당 Pull Request 담당자가 한다.

Commits on May 31, 2022	
Feat: Modify scheduler classes	bc96a83
Commits on Jun 1, 2022	
Feat: SRTF scheduler 완성화	785d60f
Fix: SRTF scheduler log 시작 시간 오류 해결	f85541e
Refactor: SRTF 구현 방식 변경	04e79d6
Chore: scheduler.py 주석 추가 및 클래스 변수 변경	3295a1a
Fix: PriorityPreemptiveRR을 PriorityRR로 변경	686a266
Chore: input2.txt process 개수 수정	1236338

Branch 내의 Commit Message 다른 팀원들이 쉽게 변경사항을 확인할 수 있도록 Fix / Feat / Refactor / Chore 등의 유형과 함께 내용상으로 Commit을 나누었다.



다른 팀원들은 Pull Request를 검토하며 코드를 리뷰한다.

3. 다시 새로운 작업을 할 때 Local Repository에서 Upstream Repository를 pull 한다.

iii. 협업도구

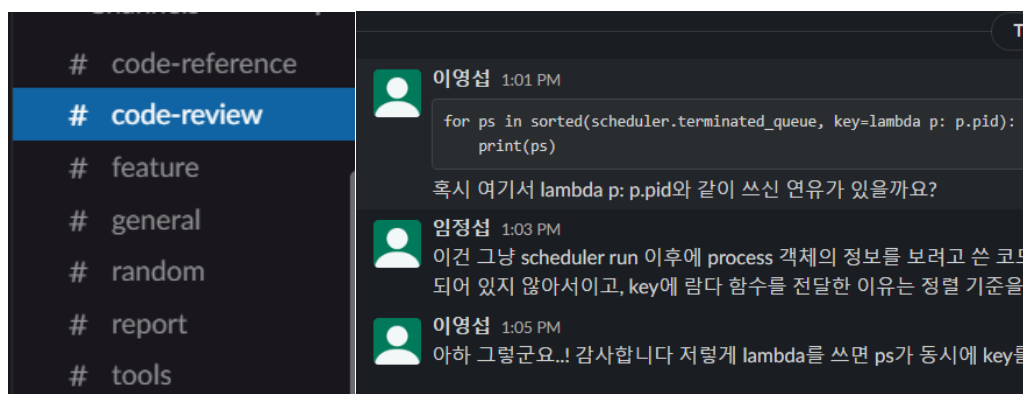
팀원 간 소통과 업무 생산성을 향상하기 위하여 어떠한 툴을 활용하였는지에 관해 짧게 소개하고자 한다. 사소하지만 도움이 되었다.

Notion(Web 기반 문서 작성), Slack(업무용 메신저), When2Meet(일정 조율), Google Calendar(공유 캘린더)

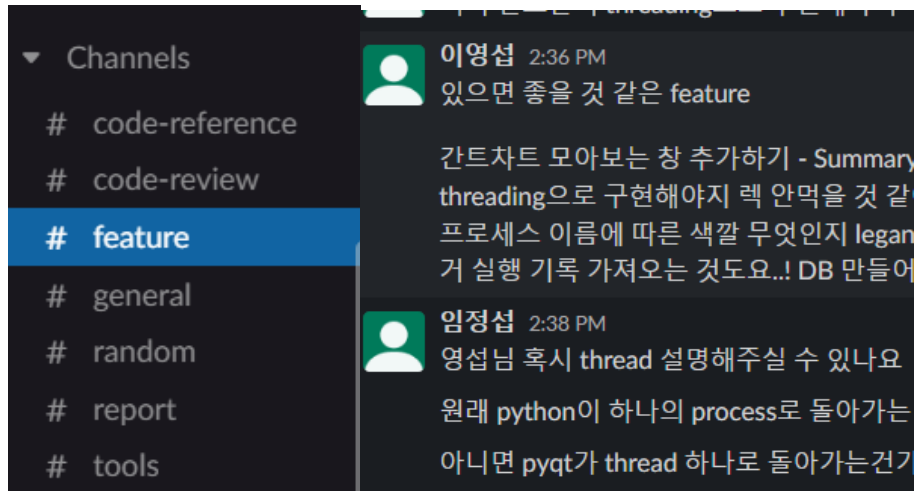
원활하게 소통하기

Slack의 channel 기능을 활용하여 대화를 주제에 따라 분리한다.

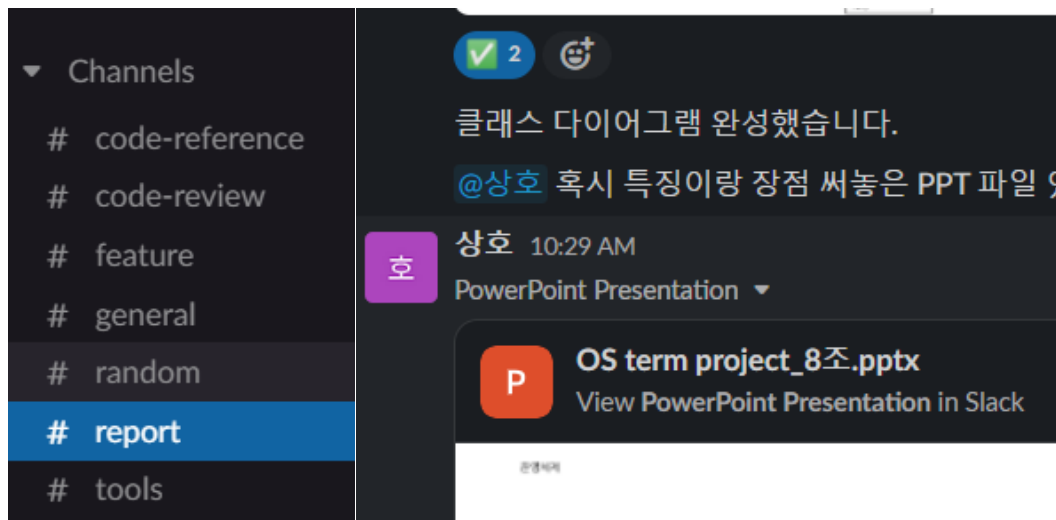
채널 별, 스레드 별로 대화를 분리하여 대화 흐름이 일관되게 유지되도록 한다.



code-review 채널에서 작성한 코드에 관해 질문한다.

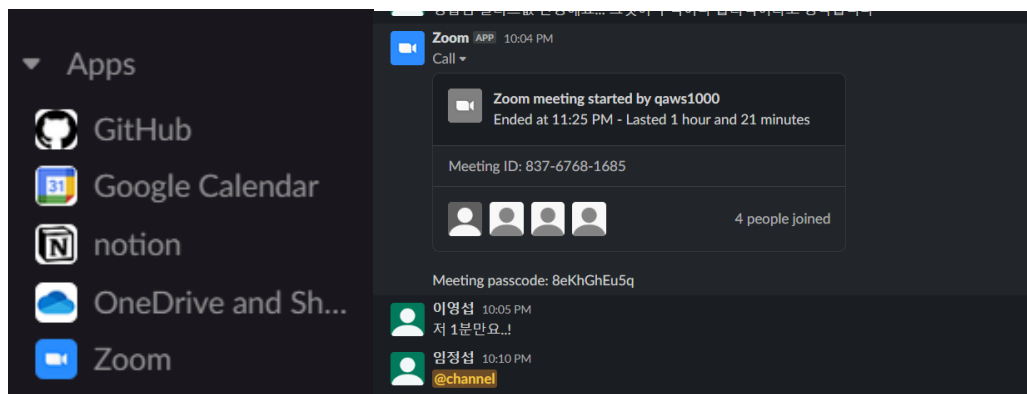


feature 채널에서 추가하면 좋은 기능에 대해 이야기한다.



report 채널에서 보고서 작성에 관해 이야기한다.

대화 중 필요 시 즉시 Zoom meeting을 열어 회의한다.



Slack integration과 단축키 명령어를 활용하면 대화 도중에도 쉽게 zoom meeting을 열 수 있다.

자료 공유 및 한 눈에 보기

업무에 필요한 정보를 팀원 전체가 공유하고 필요한 경우, 즉시 찾을 수 있도록 정리하여 두어 생산성을 증진하였다.

Notion 공유 문서의 메인화면에 업무에 필요한 핵심자료들을 올려둔다.



Add cover Add comment

OS Term Project 8조

Tasks

회의록

UI

Plan

발표 대본

Report

Whimsical

Embed Examples

The Visual Workspace | Whimsical

<https://whimsical.com/os-term-project-9pDxRUJTFYm6HEkaCXJVD8>



Google Drive

drive.google.com

<https://drive.google.com/drive/u/0/folders/0AALjhlcge-B1Uk9PVA>

Slack

Slack

<https://w22spring-os.slack.com/>

Github Repository

Github Convention

cpu-scheduling-simulator-python
jseop-lim • Updated 4 days ago

...

When2meet

OS teamproject 회의 날짜 - When2meet

<https://www.when2meet.com/?15588447-4hjcg>

Reference

프로세스 교체 결정 계산

Task 단위로 정리하여 지난 문서를 찾을 수 있도록 한다.

→ OS Term Project 8조 / 회의록

Add cover Add description

회의록

모든 회의 회의 유형별 Add view

- [6/3] 발표 및 개선사항
- [5/30] GUI 점검 및 발표 준비
- [5/19] GUI 사용 결정 및 계획서 작성
- [5/16] 계획서 작성
- [5/12] 코드 설명 및 역할 분담 3
- [5/10] 일정 계획 및 모듈 설계
- [5/6] 첫 회의

+ New

회의 단위로 정리한 모습

완료 18

진시훈

UI 제작
youngseob lee
May 21, 2022

[PyQt] 간트차트 출력
진시훈

GUI 입력 모듈
임정섭 진시훈
8

GUI 입력 모듈

Assignee

임정섭 진시훈

Status

완료

Priority

Empty

Due Date

Empty

Date Created

May 20, 2022 1:48 AM

Attachment

Empty

Project

Empty

Add a property

진시훈 May 24 (edited) 22.05.24

[진행상황]

1) pid, arrival, burst, priority 입력해서 add 버튼 누르면 model 객체의 base_process_list에 추가될 수 있도록 + 입력완료 창(표)에 추가되도록 구현.
- 4개 중 어느 하나라도 비어 있으면 에러 메시지 뜨도록 완료.

2) delete All 버튼 클릭 시 base_process_list.clear(), 입력완료 창(표)도 모두 사라지도록.

3) timeslice 입력 후 Run 버튼 클릭 시
- SRTF를 제외한 나머지 6가지 알고리즘의 results (response time, turnaround time, waiting time)이 각 프로

각각의 task를 처리할 때 작성한 문서들은 모두 그 task 페이지에 담겨 있다.

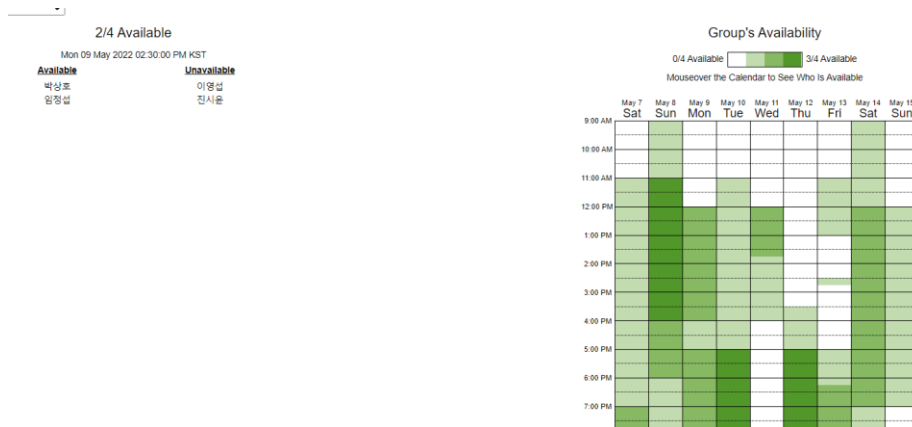
이를 통해 지난 작업 때 참고했던 자료를 바로 확인할 수 있다.

수정한 사항들은 slack integration을 활용하여 팀원들에게 알린다.



일정, 할 일 관리하기

When2Meet을 통한 팀원 간 일정 협의



서로의 일정을 구두로 확인하지 않아도 Web을 활용하여 일정을 조율할 수 있다.

Google Shared Calendar에 일정 생성

MON 25	TUE 26	WED 27	THU 28	FRI 29	SAT 30	SUN May 1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16 ● 7pm 회의	17	18	19 ● 11:30pm tmpj 내 설명	20 ● 11am 온체 팀플 회의	21	22 ospy-pyqt5 내 환경에서
23	24	25	26	27 ● 9am OS 회의	28	29
30 ● 10pm 회의(zoom)	31	Jun 1	2	3 OS 발표 및 데모 ● 11am 온체 팀플 발표 ● 10pm 회의(zoom)	4	5

회의 일정이 정해지면 Google Shared Calendar를 통해 팀원 모두 착오 없이 일정을 확인할 수 있다.

Invitation: 회의(zoom) @ Fri 2022-06-03 10pm - 11pm (KST) (subspn@gmail.com) -: * qaws1000@korea.ac.kr - creator * 진시윤 * subspn@gmail.com

New event: 회의(zoom) @ Fri 2022-06-03 10pm - 11pm (KST) (Operating System) -: * qaws1000@korea.ac.kr - creator * 진시윤 * 이영섭 * 상

New event: 온체 팀플 발표 준비 @ Fri 2022-06-03 10:30am - 12:30pm (KST) (Operating System) -: * qaws1000@korea.ac.kr - organizer * psh

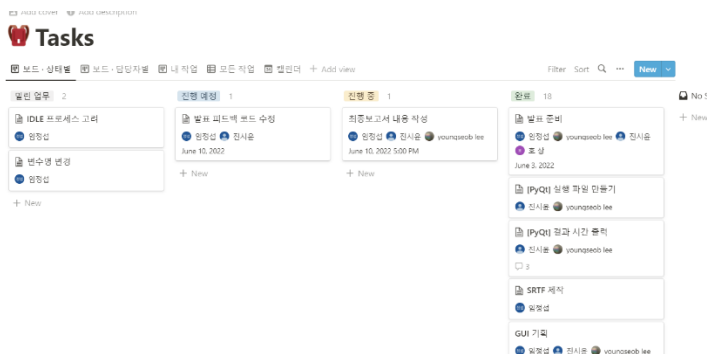
Canceled event: 온체 팀플 발표 준비 @ Fri 2022-06-03 11am - 12:30pm (KST) (subspn@gmail.com) -: * qaws1000@korea.ac.kr - organizer

Invitation: 온체 팀플 발표 준비 @ Fri 2022-06-03 11am - 12:30pm (KST) (subspn@gmail.com) -: * qaws1000@korea.ac.kr - organizer * subspn@gmail.com

Invitation: 온체 팀플 회의 @ Wed 2022-06-01 10:30pm - 11:30pm (KST) (subspn@gmail.com) -: * qaws1000@korea.ac.kr - organizer * subspn@gmail.com

메일 알림을 등록해두면 새로 생긴 일정을 이메일을 통해 리마인드할 수 있다.

Notion의 Task Board를 활용하여 프로젝트의 TO-DO를 관리한다.



진행 중인 업무와 담당자를 칸반보드로 확인한다.

II. 프로그램 소개

i. 개발언어

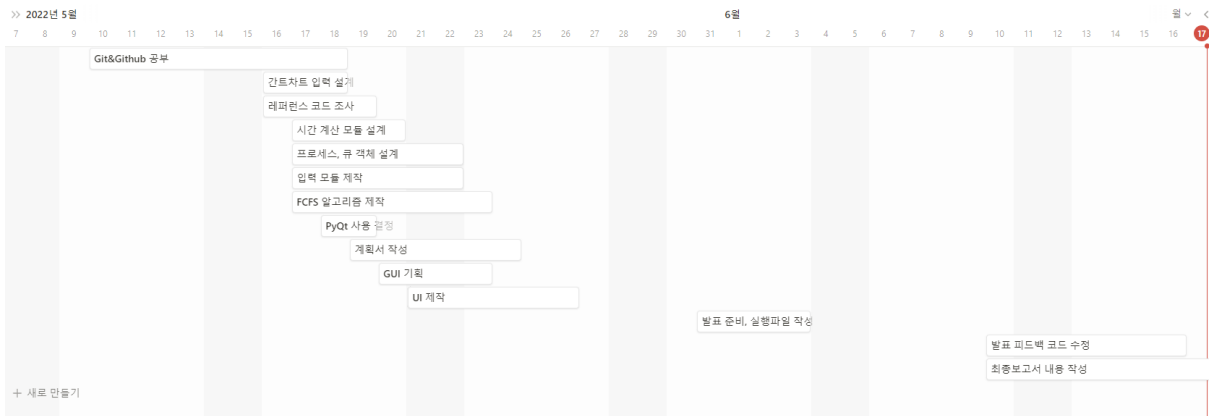
개발언어: Python

주요 프레임워크: PyQt5, plotly

PyQt5는 GUI 프레임워크로 python을 이용한 GUI 프로그래밍에 사용됨

Plotly는 시각화 패키지로서 간트차트 그리기에 사용함

ii. 개발일정



III. 프로그램 구조

i. 프로그램 설계

스케줄러: 프로세스 정보를 입력받아서 스케줄링 알고리즘을 수행하고 계산 결과를 반환하는 스케줄러 7개

간트차트: 스케줄러의 계산 결과를 바탕으로 간트차트의 이미지를 생성함

GUI 디자인: GUI의 화면 디자인하기

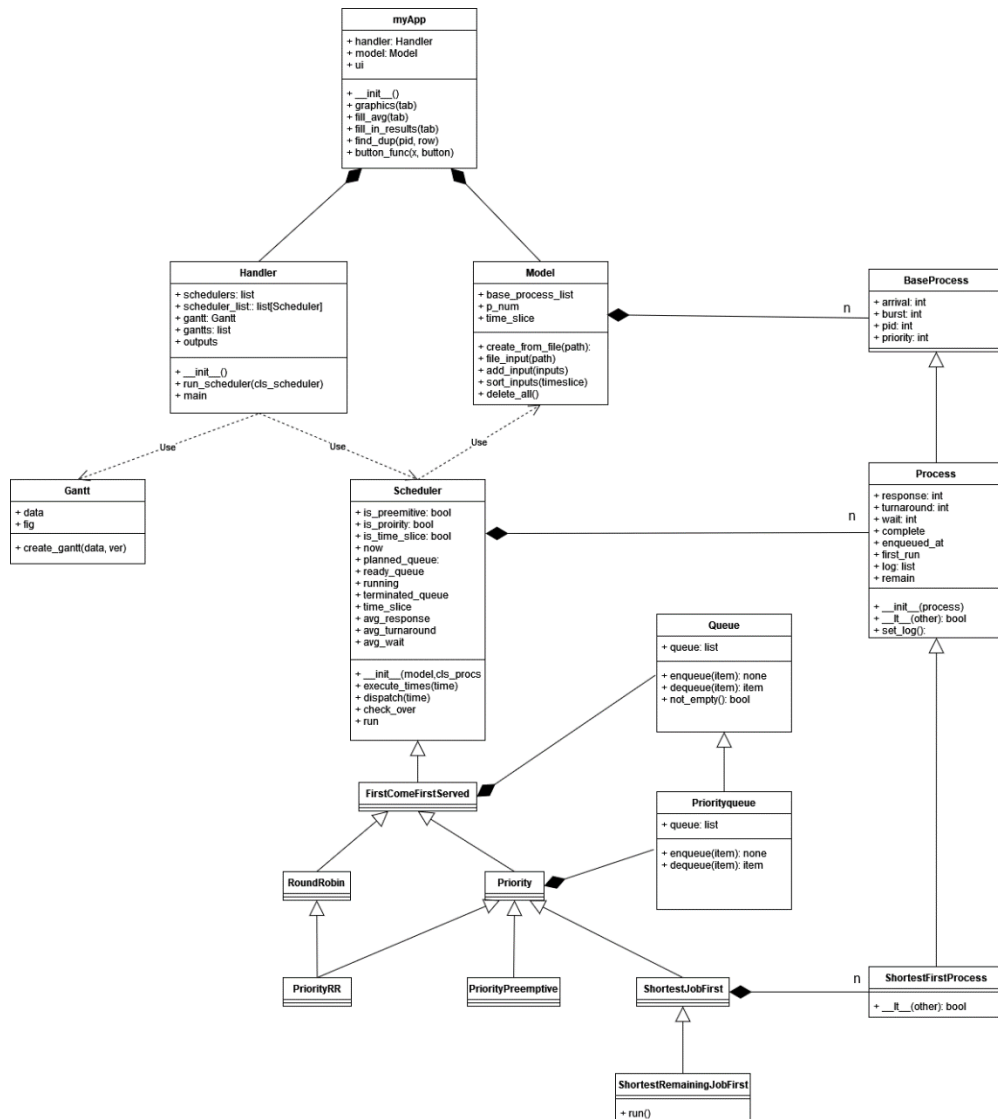
GUI 컨트롤러: 사용자가 GUI 입력창에 쓴 값들을 읽고, 스케줄러 및 간트차트 모듈을 실행하고, 다시 GUI에 출력해주는 GUI 컨트롤러

사용자가 값을 입력하면 스케줄러가 PyQt5를 통해 결과값을 계산한다.

계산한 결과값을 plotly를 통해 간트차트를 만들고 차트 이미지를 생성한다.

gui 컨트롤러가 이 모두를 수합하여 gui 화면에 출력한다

ii. 클래스 다이어그램



설계를 바탕으로, 파이썬으로 객체지향* 개념을 적용하여 프로그램을 제작했다. 구현된 결과를 클래스 다이어그램으로 시각화했다. 스케줄러를 추상화하여 일관된 방법으로 알고리즘을 프로그램에 추가할 수 있다. 또한, 스케줄러를 실행하는 과정에서 전략패턴을 사용하여, 중복된 코드 없이 간편하게 원하는 알고리즘을 선택할 수 있다.

(*) 객체지향 프로그래밍: 프로그램 구현에 필요한 객체를 파악하고 각각의 객체들의 역할이 무엇인지를 정의하여 객체들 간의 상호작용을 통해 프로그램을 만드는 것
클래스 다이어그램의 각 부분을 살펴보고 각 클래스에 관해 자세히 설명하고, 클래스 사이에 어떤 관계가 있는지 알아본다.

프로세스 정보 입력

BaseProcess는 사용자가 UI를 통해 입력하는 정보인 pid, arrival time, burst time, priority를 저장하는 자료구조이다. 나중에 스케줄러에서는 한 프로세스에 관해 이와 더

불어 remaining time, first run time, enqueued time 등을 추가로 저장하므로 BaseProcess를 상속받는 Process 객체로 프로세스 정보를 관리한다.

Model은 사용자가 입력한 모든 정보를 저장한다. 즉, BaseProcess 객체의 list와 time slice, 프로세스 개수를 필드로 갖는 객체이다.

Scheduler는 사용자의 입력을 Model 객체로 전달받아 알고리즘을 수행하는 역할이다. 생성자 메서드 __init__()에서 BaseProcess list의 원소들을 Process 객체로 변환한다. run() 메서드를 호출하면 알고리즘이 실행된다. 수행 결과는 Process 객체의 필드인 response time, turnaround time, waiting time, gantt log에 기록된다.

단일 알고리즘 수행

Scheduler는 추상클래스로 정의되어 있으며, 이를 상속받는 자식 클래스는 is_priority, is_preemptive, is_time_slice 클래스 변수를 설정하여 특성을 반영할 수 있다. configuration의 의미는 아래와 같으며, 모든 알고리즘은 configuration의 조합으로 표현할 수 있다.

- is_priority: ready queue가 우선순위 큐이면 True, FIFO 큐이면 False
- is_preemptive: 프로세스 실행 도중 우선순위가 높은 프로세스가 도착하면 교체 허용
- is_time_slice: time slice 적용

Scheduling Algorithm	is_priority	is_preemptive	is_time_slice
FCFS	X	X	X
SJF (*)	O	X	X
STRF (*)	O	O	X
RR	X	X	O
Non-preemptive Priority	O	X	X
Preemptive Priority	O	O	X
Non-preemptive Priority & RR	O	X	O
Preemptive Priority & RR	O	O	O

(*) SJF, SRTF는 ready queue 우선순위로 remaining time을 사용함. 나머지는 enqueued_at(가장 최근에 ready queue에 삽입된 시각)을 사용함

이러한 알고리즘 간의 관계를 상속 관계로 표현하여 클래스 다이어그램 하단부에 나타냈다. 실제 상속관계가 표현된 코드는 아래와 같다.

```
class FirstComeFirstServed(Scheduler):
    is_preemptive = False # 실행 중간에 프로세스 교체 허용?
    is_priority = False # ready queue가 priority queue or FIFO queue
    is_time_slice = False # time slice 적용?

class Priority(FirstComeFirstServed):
    is_priority = True

class PriorityPreemptive(Priority):
    is_preemptive = True

class RoundRobin(FirstComeFirstServed):
    is_time_slice = True

class PriorityRR(Priority, RoundRobin):
    pass

class ShortestJobFirst(Priority):
    process_class = ShortestFirstProcess

class ShortestRemainingTimeFirst(ShortestJobFirst):
    is_preemptive = True

def run(self):
```

전체 스케줄러 실행

Handler 객체는 전체 스케줄러의 알고리즘을 수행하고 간트차트 이미지를 생성하여 저장한다.

Handler에서는 모든 실제 Scheduler 클래스를 리스트로 소유한다. run_scheduler()는 하나의 스케줄러를 실행하고 결과를 딕셔너리로 구조화하여 반환하는 메서드이다. main()은 내부에서 모든 스케줄러에 대한 run_scheduler()를 호출하여 전체 스케줄러를 실행하고 간트차트 이미지를 생성하여 저장한다.

Run_scheduler()는 Scheduler 클래스를 매개변수로 전달받아 해당 스케줄러의 run() 메서드를 실행한다. Scheduler 클래스에 FCFS, RR 등이 존재하므로 매개변수를 조절하여 실행하고자 하는 알고리즘을 쉽게 전환가능하다. 이처럼 스케줄러 실행 과정에 전략패턴을 적용했다.

GUI 컨트롤러의 동작

GUI 컨트롤러의 모든 기능은 myApp 클래스의 메서드로 구현된다. myApp 객체는 Handler 객체와 Model 객체를 소유한다. 그리하여 GUI 화면에 사용자가 입력한 프로세스 정보와 time slice를 Model 객체에 저장하고, 사용자가 실행 버튼을 누르면

Handler 객체의 main() 메서드를 호출하고 그 결과를 GUI 화면에 출력한다.

iii. 자료구조 / 주요변수

스케줄러 코드 설명

```
class Scheduler:
    is_preemptive = None # 실행 중간에 프로세스 교체 허용?
    is_priority = None # ready queue 가 priority queue or FIFO
    queue
    is_time_slice = None # time slice 적용?
    process_class = Process
    idle = Process(BaseProcess(pid='idle', arrival=float('inf'),
    burst=float('inf'), priority=float('inf')))

    def __init__(self, model: Model):
        self.planned_queue = list(map(self.process_class,
    model.base_process_list)) # 아직 도착하지 않은 프로세스
        self.ready_queue = PriorityQueue() if self.is_priority else
    Queue() # 도착했지만 실행 중이 아닌 프로세스
        self.terminated_queue = [] # 실행이 끝나 종료된 프로세스
        self.running = None # 실행 중인 프로세스
        self.time_slice = model.time_slice if self.is_time_slice
    else max(p.burst for p in model.base_process_list)

        try:
            self.running = self.planned_queue.pop(0)
            self.now = self.running.arrival
        except IndexError:
            raise IndexError('프로세스를 하나 이상 입력하세요.')

    def execute_times(self, time):
        """
        시간 증가
        """
        self.now += time
        self.running.remain -= time

    def dispatch(self, time=None):
        """
        context switching
        = 실행 중인 프로세스를 ready queue 혹은 terminated queue 에
        삽입하고
        ready queue 에서 프로세스를 하나 빼서 실행시킴

        :param time: 입력 시, 시간 증가시키고 간트차트에 기록
        :return:
        """
        if time:
```

```

        print(self.now, self.running)
        self.running.set_log(self.now, time) # gantt chart
        self.execute_times(time)

    if self.running.remain > 0:
        self.running.enqueue_at = self.now
        self.ready_queue.enqueue(self.running)
    else:
        self.terminated_queue.append(self.running)
        self.running.complete = self.now
        self.running = self.ready_queue.dequeue() if
self.ready_queue.not_empty() else None

    def check_over(self):
        """
        :return: 모든 프로세스가 실행되었다면 False, 아니면 True
        """

        return self.running or self.ready_queue.not_empty() or
self.planned_queue

    def run(self):
        """
        프로세스 실행하고 process 객체의 필드(first_run, complete,
log - gantt chart 입력용) 업데이트
        """

        while self.check_over():
            # save first_run time
            if self.running.first_run is None:
                self.running.first_run = self.now

            next_dispatch_time = self.running.remain
            if self.is_time_slice and self.time_slice <
self.running.remain:
                next_dispatch_time = self.time_slice

            if self.planned_queue:
                next_arrival_time = self.planned_queue[0].arrival -
self.now

                if next_arrival_time <= next_dispatch_time:
                    # arrive
                    new_process = self.planned_queue.pop(0)
                    self.ready_queue.enqueue(new_process)
                    # preempt by priority
                    if self.is_priority and next_arrival_time == 0
and new_process < self.running:
                        self.dispatch()
                        if self.running.first_run == self.now:
                            self.running.first_run = None
                        elif self.is_preemptive and new_process <
self.running:

```

```
        self.dispatch(next_arrival_time)
        continue

        self.dispatch(next_dispatch_time)

    print(self.now, self.running)
```

queues; 프로세스 저장

프로세스는 총 4가지 상태로 존재하며, 상태에 따라 각각 다른 큐에 저장된다.

- planned_queue: 아직 도착하지 않은 프로세스
- ready_queue: 도착했지만 실행 중이 아닌 프로세스
- running: 실행 중인 프로세스
- terminated_queue: 실행이 완료된 프로세스

planned_queue, terminated_queue는 FIFO 방식으로 삽입 삭제되며 python list로 구현했다.

ready_queue는 FIFO 방식 혹은 우선순위 적용 시 우선순위 큐 방식으로 동작한다.

running은 단일 프로세스를 지정한다.

따라서 프로세스의 생애 주기는 다음과 같다.

1. 초기화: planned_queue에 저장된다.
2. 도착: planned_queue에서 ready_queue로 이동한다.
3. 실행: ready_queue에서 running으로 이동한다.
4. 선점됨(시간초과 포함): running에서 ready_queue로 이동한다.
5. 실행
6. 종료: running에서 terminated_queue로 이동한다.

비선점 알고리즘(is_preemptive=False)에서는 4, 5가 생략된다.

run(); 프로세스 실행

외부에서 호출되는 메서드는 오직 run()이다. Scheduler를 상속받는 FCFS, RR 등의 스케줄러는 boolean 형태의 in_priority, is_preemptive, is_time_slice 클래스 변수를 반드시 정의해야 한다.

run()은 세 클래스 변수에 관한 조건문에 따라 분기하여 해당 기능을 실행하거나 안하게 된다.

time slice의 적용 여부, 비선점/선점에 따라 프로세스가 교체될 수 있는 경우를 정리했다. 즉, is_time_slice, is_preemptive가 True이면 각 경우에 해당하는 프로세스 교체 결정 로직을 실행하는 것이다.

- (time slice X)
비선점(FCFS): 무조건 현재 실행 중인 프로세스의 burst time이 지날 때 프로세스 교체 선점
 - 우선순위: 새로운 프로세스가 도착하면 프로세스 교체 검사
 - SRTF: 새로운 프로세스가 도착하면 실행 중인 프로세스의 remain time 계산하고 새로운 프로세스의 burst time과 비교하여 프로세스 교체 검사
- (time slice O)
RR: 현재 실행 중인 프로세스의 remain time과 time slice 중 짧은 것만큼의 시간이 지나면 프로세스 교체
RR&선점 우선순위: $\min\{\text{현재 프로세스의 remain time, time slice}\}$ 지나면 프로세스 교체, 새로운 프로세스 들어오면 우선순위 비교하여 프로세스 교체 검사
- RR&우선순위 psuedo code

```
dispatch_time = min(now+run_remain, now+time_slice)

if arrival_list[0] < dispatch_time:
    if check_priority():
        dispatch()
else:
    dispatch()
```

이렇게 교체 가능성이 있는 시각만 고려하여 만약 교체가 발생한다면 dispatch() 메서드를 호출하고 현재시각(self.now)를 늘린다. (현재시각 증가는 dispatch() 메서드 내부에서 실행됨) 더 이상 실행 가능한 프로세스가 남아있지 않다면 실행을 종료한다.

프로세스 코드 설명

```
from dataclasses import dataclass, field

@dataclass
class BaseProcess:
    pid: str
    arrival: int
    burst: int
    priority: int

@dataclass
class Process(BaseProcess):
    remain: str
```

```

complete: int
first_run: int
enqueued_at: int
log: list = field(repr=False)

def __init__(self, process: BaseProcess):
    super().__init__(process.pid, process.arrival,
process.burst, process.priority)
    self.remain = self.burst
    self.complete = None
    self.first_run = None
    self.enqueued_at = self.arrival
    self.log = []

def __lt__(self, other):
    return (self.priority, self.enqueued_at) < (other.priority,
other.enqueued_at)

@property
def response(self):
    return None if self.first_run is None else self.first_run -
self.arrival

@property
def turnaround(self):
    return None if self.complete is None else self.complete -
self.arrival

@property
def wait(self):
    return None if self.turnaround is None else self.turnaround
- self.burst

def set_log(self, start, time):
    self.log.append([start, start + time])

class ShortestFirstProcess(Process):
    def __lt__(self, other):
        return (self.remain, self.enqueued_at) < (other.remain,
other.enqueued_at)

```

프로세스를 저장하는 자료구조와 프로세스의 response, turnaround, wait time을 계산하는 방법을 알아본다.

프로세스 자료구조

사용자가 process에 관해 입력하는 정보인 pid, 도착시간, 실행시간은 **BaseProcess** 객체에 저장된다.

스케줄러 구동 시 각 프로세스는 remain time 등을 추가로 저장해야 한다. Scheduler 객체는 BaseProcess를 입력으로 받아서 **Process** 객체를 생성하고, Process 객체의 리스트로 프로세스들을 관리한다.

Process 클래스는 프로세스 간 우선순위 비교 함수 `_lt_()`를 정의한다. `ready_queue`가 우선순위 큐 방식으로 동작할 때 사용된다. SJF, SRTF는 사용자 입력 우선순위가 아닌 remain time을 우선순위로 사용하므로 `_lt_()`를 오버라이딩하는 **ShortestFirstProcess**를 따로 정의하여 사용한다.

시간 계산

손으로 response, turnaround, wait time를 구할 때는 간트차트를 그린 후에 분할된 실행 시간들을 직접 더해가지만, 수식을 사용하여 이를 쉽게 계산할 수 있다. (property 메서드 정의 참고) 이를 위해서는 각 프로세스의 첫 실행 시작 시각과 종료 시각을 알아야 한다.

Scheduler 클래스의 `run()` 메서드가 호출되어 프로세스가 실행되는 동안 `run()` 메서드에서는 Process 객체의 `first_run`, `complete` 필드에 접근하여 값을 갱신한다. `first_run`은 해당 프로세스가 처음 실행될 때 시작 시각이다. `complete`는 해당 프로세스가 종료될 때 시각을 기록한다. 이 정보를 바탕으로 response, turnaround, wait time이 계산되며, 각각이 property 메서드로 정의되어 있다.

간트차트 코드 설명

```
import plotly.figure_factory as ff
import plotly.io as pio
from plotly.offline import plot
import plotly.express as px
import os
from html2image import Html2Image

import urllib.request as req

hti = Html2Image()
path = os.path.dirname(os.path.abspath(__file__))

class Gantt:
    def __init__(self):
        self.data = []

    def create_gantt(self, data, ver):
```

```

"""
    scheduler log 를 data 로 입력받아 image 파일을 생성
    ver: scheduler 이름
"""
self.data = data
#labels = self.data.keys()
df = []
annots = []
ticks = []
for pid, time in self.data.items():
    d = {}
    annot_d = {}
    tick = {}
    if isinstance(time, list):
        for nested in time:
            d = dict(Task = "Gantt", Subtask = pid, Start =
nested[0], Finish = nested[1], Resource= pid)
            df.append(d)
            d = {}
            annot_d = dict(x=(nested[0]+nested[1])/2, text =
pid, showarrow=False, font = dict(color = 'black', size =40))
            annots.append(annot_d)
            annot_d = {}
            tick = dict(x=nested[1], y = -0.5, text =
str(nested[1]), showarrow = False, font = dict(color='red',size =
32 ) )
            annots.append(tick)
            tick = {}

        else :
            d = dict(Task = "Gantt", Subtask = pid, Start =
time[0], Finish = time[1], Resource = pid)
            df.append(d)
            annots.append(annot_d)
            annots.append(tick)

            annots.append(dict(x=0, y = -0.5, text = '0', showarrow =
False, font = dict(color='red',size = 32 )))
            self.fig = ff.create_gantt(df, index_col = 'Subtask',
group_tasks = True)
            self.fig.update_layout(
                xaxis_type = 'linear',
                showlegend = True,
                xaxis = dict(
                    showticklabels = False,
                    showgrid = False,
                    # gridcolor = 'gray',
                    # linecolor = 'gray',
                )
            )

```



```

self.fig.layout.xaxis['tickfont'] = {'size': 40}

self.fig['layout'].update(
    annotations= annots
)

#self.fig.show()
if not os.path.exists(os.path.join(path, 'html')):
    os.makedirs(os.path.join(path, 'html'))

filepath = os.path.join(path, 'html/image_' + ver + '.html')
pio.write_html(self.fig, filepath)
filename = os.path.join(path, 'image/' + ver + '.png')

with open(filepath) as f:
    if ver == 'FCFS':
        hti.screenshot(f.read(), save_as = 'image_FCFS.png')
        f.close()
    elif ver == 'PP' :
        hti.screenshot(f.read(), save_as = 'image_PP.png')
        f.close()
    elif ver == 'PRR' :
        hti.screenshot(f.read(), save_as = 'image_PRR.png')
        f.close()
    elif ver == 'Priority' :
        hti.screenshot(f.read(), save_as =
'image_Priority.png')
        f.close()
    elif ver == 'RR' :
        hti.screenshot(f.read(), save_as = 'image_RR.png')
        f.close()
    elif ver == 'SJF' :
        hti.screenshot(f.read(), save_as = 'image_SJF.png')
        f.close()
    elif ver == 'SRTF':
        hti.screenshot(f.read(), save_as = 'image_SRTF.png')
        f.close()

```

gantt; 간트차트 이미지 저장

간트차트는 스케줄러로부터 반환받은 값들 중 gantt_data의 값을 받아서 만들어진다. 따라서, 스케줄러 실행 이후에 handler의 main함수에서 만들어지는데, create_gantt(self, data, ver)로 만들어진다.

create_gantt(self, data, ver): 간트차트를 그릴 준비를 해 줌

(*) data: 간트차트를 만들기 위해 필요한 간트차트 데이터를 의미한다.

handler로부터 넘겨 받은 gantt_data는 {pid: [시작 시간, 실행 기간]}의 dictionary 형태로 되어 있다. 간트 차트를 만들기 위해, plotly.figure_factory.create_gantt 함수를 쓴다.

이 함수를 쓰기 위한 input data를 정제해야 한다. Gantt_data가 딕셔너리 형태로 저장되어 있지만, 하나의 프로세스가 여러 번에 걸쳐서 실행될 수도 있어, 즉, value 값이 nested list 형태일 수도 있기 때문이다. 따라서, 딕셔너리에서 key값에 대한 value값이 list이나 아니냐에 따라서 정제를 해주고, 그렇게 정제된 데이터를 plotly.figure_factory.create_gantt의 input으로 주어 간트 차트 이미지를 생성한다.

이렇게 생성된 이미지를 plotly.io.write_html을 통해서 html파일로 만들어 준다.

html로 바뀐 후, 이를 png로 변환해주는 과정을 거치게 된다.

이 때, ver으로 받은 이 스케줄러의 이름을 딴 이미지 파일을 생성할 수 있게 한다. 그 결과, 간트차트 이미지가 생성되며, 이렇게 생성된 간트차트 이미지는 추후 gui에서 불러온다.

iv. 알고리즘 핵심코드

GUI 소스코드 설명

```
import sys
from PyQt5.QtWidgets import *
from PyQt5 import uic
from PyQt5.QtGui import *
import os
from PyQt5.QtCore import pyqtSlot
from input import Model
from handler import Handler
from html2image import Html2Image

ui_path = os.path.dirname(os.path.abspath(__file__))
form_class = uic.loadUiType(os.path.join(ui_path,
"mainwindow.ui"))[0]

hti = Html2Image()

class MyApp(QMainWindow, form_class):
    def __init__(self):
        super().__init__()
        self.ui = uic.loadUi(os.path.join(ui_path,
"mainwindow.ui"), self)
        #self.setupUi(self)
        self.setWindowTitle("My Scheduler")
        self.model = Model()
        self.handler = Handler()
```

```

self.handler.model = self.model
lineEdit_list = [self.ui.lineEdit_arrival,
                  self.ui.lineEdit_burst,
                  self.ui.lineEdit_pid,
                  self.ui.lineEdit_priority,
                  self.ui.lineEdit_timeslice]

#validation
#1) arrival, burst, timeslice : double 로 제한
#2) priority : int 로 제한
intValidator = QIntValidator(0,100)
floatValidator = QDoubleValidator(0.0, 100.0, 2)

lineEdit_list[0].setValidator(floatValidator)
lineEdit_list[1].setValidator(floatValidator)
lineEdit_list[3].setValidator(intValidator)
lineEdit_list[4].setValidator(floatValidator)
#self.ui.lineEdit_priority.setInputMask("00") #99 개까지만?
#for lineEdit in lineEdit_list:
#    lineEdit.setValidator() # #pid 는 string, arrival, burst
#    등은 숫자로 제한.
btn_list = [self.ui.pushButton_add,
             self.ui.pushButton_run,
             self.ui.pushButton_deleteall]

for btn in btn_list:
    btn.clicked.connect(lambda x, button = btn:
self.button_func(x, button))

@pyqtSlot()
def graphics(self, tab):
    scene = QGraphicsScene()
    #pname = 'image_'+tab+'.png'
    pixmap = QPixmap('image_'+tab+'.png')
    width = self.ui.gantt_fcfs.size().width()
    pixmap = pixmap.scaledToWidth(width)

    if tab == 'FCFS':
        view =self.ui.gantt_fcfs
    elif tab == 'Priority' :
        view = self.ui.gantt_priority
    elif tab == 'RR' :
        view = self.ui.gantt_rr
    elif tab == 'PP' :
        view = self.ui.gantt_pp
    elif tab == 'PRR' :
        view = self.ui.gantt_pprr
    elif tab == 'SJF':
        view = self.ui.gantt_sjf

```

```

elif tab == 'SRTF' :
    view = self.ui.gantt_srtf
    scene.addPixmap(pixmap)
    view.setScene(scene)
    view.show()

@pyqtSlot()
def fill_avg(self, tab):
    if tab == 'FCFS':

        avg = self.handler.outputs[0][4] # FCFS' avg times
        print('avg:', avg)
        times = avg.keys()
        for i, time in enumerate(times):
            self.ui.avg_fcfs.setItem(0, i,
QTableWidgetItem(str(avg[time])))

    elif tab == 'RR':
        avg= self.handler.outputs[1][4] # RR's avg times
        times =avg.keys()
        for i, time in enumerate(times):
            self.ui.avg_rr.setItem(0, i,
QTableWidgetItem(str(avg[time])))

    elif tab == 'Priority':
        avg= self.handler.outputs[2][4] #Priority's avg times
        times =avg.keys()
        for i, time in enumerate(times):
            self.ui.avg_priority.setItem(0, i,
QTableWidgetItem(str(avg[time])))

    elif tab == 'PP':
        avg= self.handler.outputs[3][4] #PP's avg times
        times =avg.keys()
        for i, time in enumerate(times):
            self.ui.avg_pp.setItem(0, i,
QTableWidgetItem(str(avg[time])))

    elif tab == 'PRR':
        avg= self.handler.outputs[4][4] #PPRR's avg times
        times =avg.keys()
        for i, time in enumerate(times):
            self.ui.avg_pprr.setItem(0, i,
QTableWidgetItem(str(avg[time])))

    elif tab == 'SJF':
        avg= self.handler.outputs[5][4] #SJF's avg times
        times =avg.keys()
        for i, time in enumerate(times):
            self.ui.avg_sjf.setItem(0, i,

```

```

QTableWidgetItem(str(avg[time])))

        elif tab == 'SRTF':
            avg= self.handler.outputs[6][4] #SRTF's avg times
            times =avg.keys()
            for i, time in enumerate(times):
                self.ui.avg_srtf.setItem(0, i,
QTableWidgetItem(str(avg[time])))

    @pyqtSlot()

    def fill_in_results(self, tab):
        if tab == 'FCFS':
            response = self.handler.outputs[0][0] # FCFS' waiting
time
            turnaround = self.handler.outputs[0][1] # FCFS'
turnaround time
            waiting = self.handler.outputs[0][2] # FCFS' waiting
time

            self.ui.results_fcfs.setRowCount(len(response))
            pid = sorted(response.keys())

            for i, id in enumerate(pid):
                self.ui.results_fcfs.setItem(i, 0,
QTableWidgetItem(id))

            self.ui.results_fcfs.setItem(i,1,QTableWidgetItem(str(response[id]))
            )

            self.ui.results_fcfs.setItem(i,2,QTableWidgetItem(str(turnaround[id]
            )))

            self.ui.results_fcfs.setItem(i,3,QTableWidgetItem(str(waiting[id])))

        elif tab == 'RR':
            response = self.handler.outputs[1][0] # RR's waiting time
            turnaround = self.handler.outputs[1][1] # RR's turnaround
time
            waiting = self.handler.outputs[1][2] # RR's waiting time

            self.ui.results_rr.setRowCount(len(response))
            pid = sorted(response.keys())

            for i, id in enumerate(pid):
                self.ui.results_rr.setItem(i, 0,
QTableWidgetItem(id))

            self.ui.results_rr.setItem(i,1,QTableWidgetItem(str(response[id])))

```

```

self.ui.results_rr.setItem(i,2,QTableWidgetItem(str(turnaround[id]))
)

self.ui.results_rr.setItem(i,3,QTableWidgetItem(str(waiting[id])))

    elif tab == 'Priority':
        response = self.handler.outputs[2][0] # Priority's
waiting time
        turnaround = self.handler.outputs[2][1] # Priority's
turnaround time
        waiting = self.handler.outputs[2][2] # Priority's
waiting time

        self.ui.results_priority.setRowCount(len(response))
        pid = sorted(response.keys())

        for i, id in enumerate(pid):
            self.ui.results_priority.setItem(i, 0,
QTableWidgetItem(id))

self.ui.results_priority.setItem(i,1,QTableWidgetItem(str(response[i
d])))

self.ui.results_priority.setItem(i,2,QTableWidgetItem(str(turnaround
[id])))

self.ui.results_priority.setItem(i,3,QTableWidgetItem(str(waiting[id
])))

    elif tab == 'PP':
        response = self.handler.outputs[3][0] # PP's waiting time
turnaround = self.handler.outputs[3][1] # PP's turnaround
time
        waiting = self.handler.outputs[3][2] # PP's waiting time
        self.ui.results_pp.setRowCount(len(response))
        pid = sorted(response.keys())

        for i, id in enumerate(pid):
            self.ui.results_pp.setItem(i, 0,
QTableWidgetItem(id))

self.ui.results_pp.setItem(i,1,QTableWidgetItem(str(response[id])))

self.ui.results_pp.setItem(i,2,QTableWidgetItem(str(turnaround[id]))
)

self.ui.results_pp.setItem(i,3,QTableWidgetItem(str(waiting[id])))

```

```

        elif tab == 'PRR':
            response = self.handler.outputs[4][0] # PRR's waiting
time
            turnaround = self.handler.outputs[4][1] # PRR's
turnaround time
            waiting = self.handler.outputs[4][2] # PRR's waiting
time

            self.ui.results_pprr.setRowCount(len(response))
            pid = sorted(response.keys())

            for i, id in enumerate(pid):
                self.ui.results_pprr.setItem(i, 0,
QTableWidgetItem(id))

            self.ui.results_pprr.setItem(i,1,QTableWidgetItem(str(response[id]))
            )

            self.ui.results_pprr.setItem(i,2,QTableWidgetItem(str(turnaround[id]
            )))

            self.ui.results_pprr.setItem(i,3,QTableWidgetItem(str(waiting[id])))

        elif tab == 'SJF':
            response = self.handler.outputs[5][0] # SJF's waiting
time
            turnaround = self.handler.outputs[5][1] # SJF's
turnaround time
            waiting = self.handler.outputs[5][2] # SJF's waiting
time

            self.ui.results_sjf.setRowCount(len(response))
            pid = sorted(response.keys())

            for i, id in enumerate(pid):
                self.ui.results_sjf.setItem(i, 0,
QTableWidgetItem(id))

            self.ui.results_sjf.setItem(i,1,QTableWidgetItem(str(response[id])))

            self.ui.results_sjf.setItem(i,2,QTableWidgetItem(str(turnaround[id])
            ))

            self.ui.results_sjf.setItem(i,3,QTableWidgetItem(str(waiting[id])))

        elif tab == 'SRTF':
            response = self.handler.outputs[6][0] # SRTF's waiting
time
            turnaround = self.handler.outputs[6][1] # SRTF's
turnaround time

```

```

        waiting = self.handler.outputs[6][2] # SRTF's waiting
time

        self.ui.results_srtf.setRowCount(len(response))
        pid = sorted(response.keys())

        for i, id in enumerate(pid):
            self.ui.results_srtf.setItem(i, 0,
QTableWidgetItem(id))

        self.ui.results_srtf.setItem(i,1,QTableWidgetItem(str(response[id]))
        )

        self.ui.results_srtf.setItem(i,2,QTableWidgetItem(str(turnaround[id]
        )))

        self.ui.results_srtf.setItem(i,3,QTableWidgetItem(str(waiting[id])))

    @pyqtSlot()
    def find_dup(self, pid, row):
        table = self.ui.tableWidget_processes
        for i in range(row):
            item = QTableWidgetItem()
            item = table.item(i,0).text()
            if pid == item:
                return False
        return True

    @pyqtSlot()
    def button_func(self, x, button):
        # 없는데 확인 validation
        exp = button.text()
        if(exp == 'add'):
            input1 = self.ui.lineEdit_pid.text()
            input2 = self.ui.lineEdit_arrival.text()
            input3 = self.ui.lineEdit_burst.text()
            input4 = self.ui.lineEdit_priority.text()

            ##validation!! 비어있는 것이 있는가?
            if (not input1 or not input2 or not input3 or not
input4):
                QMessageBox.warning(self, 'ERROR', 'You missed
something!\nPlease Enter ALL')
                return
            # validation!! 옳은 형식인가? (priority : int, pid :
string, arrival : float, burst : float)
            # init 에서 수행

            # validation Pid 같은거 확인.

```

```

        inputs = [input1, input2, input3, input4]

        #inputs 넘기기?
        #표에 저장하고 한번에 넘기기?
        row = self.ui.tableWidget_processes.rowCount()

        if(row!=0 and self.find_dup(input1, row)==False):
            QMessageBox.warning(self, 'ERROR', 'Duplicate PID!')
            return

        self.ui.tableWidget_processes.setRowCount(row+1)

        for j in range(4):
            item = QTableWidgetItem()
            text = inputs[j]
            item.setText(text)

            self.ui.tableWidget_processes.setItem(row, j, item)

        append_row = input1 + " " + input2 + " " + input3 + " " +
input4

        self.model.add_input(append_row)

        elif(exp == 'Delete All'):
            self.ui.tableWidget_processes.clear()
            self.ui.tableWidget_processes.setRowCount(0)
            self.model.base_process_list.clear()
            hlabels = ['pid', 'arrival time', 'burst time',
'priority']

        self.ui.tableWidget_processes.setHorizontalHeaderLabels(hlabels)

        elif(exp == 'Run'):
            if(len(self.model.base_process_list)<=0) :
                QMessageBox.warning(self, "ERROR", "No entry! Please
ENTER the inputs")
                return

            timeslice = self.ui.lineEdit_timeslice.text()
            if not timeslice:
                QMessageBox.warning(self, "ERROR", "You missed
timeslice!\nPlease enter TIMESLICE")
                return

            else:
                self.model.sort_inputs(timeslice)

                self.handler.main()

```

```

        for scheduler in self.handler.schedulers:
            self.fill_in_results(scheduler)

        for scheduler in self.handler.schedulers:
            self.graphics(scheduler)

        for scheduler in self.handler.schedulers:
            self.fill_avg(scheduler)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = myApp()
    ex.show()
    sys.exit(app.exec_())

```

def __init__(self):

- 처음 gui를 실행하면, 상대경로를 통해, mainwindow.ui를 불러온다.
- 이 클래스의 handler와 model을 선언한다.
- 입력 칸이 pid, arrival time, burst time, priority, timeslice 5가지가 있다.
입력 5가지에 대해서 validator를 설정해주어, priority는 자연수꼴로, arrival time, burst time, timeslice는 실수형으로 입력 받을 수 있도록 설정해준다. (pid는 string)
- 버튼이 Run, Delete All, Add 3가지인데, 각각에 대해서 button_func함수를 통해 button 클릭 시의 function을 설정해준다.

def button_func(self, x, button):

버튼이 총 3개 존재: Add, Run, Delete All

각 버튼들에 대해 각기 다른 액션을 취한다.

① Add인 경우

입력된 pid, arrival time, burst time, priority를 읽어 들어, 각종 validation을 거친다. (ex. 비어 있는 것, pid duplicate 여부 확인)

validation을 마치면 add를 눌렀을 때, 입력 완료된 process에 대한 정보를 저장하는 테이블이 업데이트되며, handler의 model의 base_process_list에 해당 정보를 추가한다.

② Delete All인 경우

process에 대한 정보를 저장하는 테이블을 clear하고, handler의 model의 base_process_list도 clear한다.

③ Run인 경우

handler.model.base_process_list가 비어있는지 여부를 확인해 실행 가능 여부를 확인한다.

arrival time 기준으로 base_process_list를 정렬하고, handler.main()을 호출해 스케줄링을 시작한다.

그 후의 결과를 여러 메서드를 이용해 채워줌: **fill_in_results**, **graphics**, **fill_avg**

def graphics(self, tab):

handler.main()을 수행하면, 스케줄링이 됨과 동시에, 간트차트를 생성하여 이미지 파일로 저장하게 된다. Graphics는 각 tab에 따라 (gantt chart 만들 때는 ver였던 인자) tab의 이름에 맞는 저장된 이미지를 불러온다.

def fill_avg(self, tab):

각 탭을 클릭했을 때, 그 탭에 위치한 테이블에 평균 반환시간, 평균 응답 시간, 평균 대기 시간을 출력할 수 있게 해준다.

총 7가지 스케줄러의 index를 i라고 한다면. 평균 시간에 대한 정보는 self.handler.outputs[i][0] (for i=0 to 6)에 딕셔너리 형태로 저장되어 있어, 각 값들을 불러와 테이블 위젯의 각 셀에 아이템으로 출력해준다.

def fill_in_results(self,tab):

각 프로세스의 반환 시간, 응답 시간, 대기 시간은 각각 self.handler.outputs[i][0], self.handler.outputs[i][1], self.handler.outputs[i][2]에 딕셔너리 형태로 저장되어 있다.

따라서, Pid 기준으로 정렬하되, 각 tab에 맞는 테이블 위젯의 셀을 채워 넣어 줄 수 있도록 해준다.

def find_dup(self, pid, row):

각 row를 탐색하며, 중복되는 pid가 있는지 여부를 확인한다.

v. UI 특징

GUI 제공: 사용자가 눈으로 입력 및 출력을 함

UX: 스케줄링 계산은 모두 알고리즘에 대해 한꺼번에 진행됨, 출력화면에 탭을 적용하여 여러 알고리즘 간의 출력 전환이 간편함

Validation: 잘못된 입력에 대해 거부 및 경고 메시지 출력

GUI/파일 입력 모두 가능(main이 2개)

실행 첫 화면

- 36 -

pid, arrival, burst, priority 입력

The screenshot shows the 'My Scheduler' application window. The 'Priority' tab is selected. The interface includes a 'Timeslice' input field, 'Run' and 'Delete All' buttons, and three main display areas. The top-left area is empty. The bottom-left area contains a table with headers: pid, response time, urnaround time, and waiting time. The bottom-right area contains a table with headers: pid, arrival time, burst time, and priority. Below this table are input fields for each header: P0, 0, 4, and 2. An 'add' button is located to the right of these input fields.

pid	arrival time	burst time	priority
P0	0	4	2

pid, arrival, burst, priority 입력 후 add 버튼 클릭 시

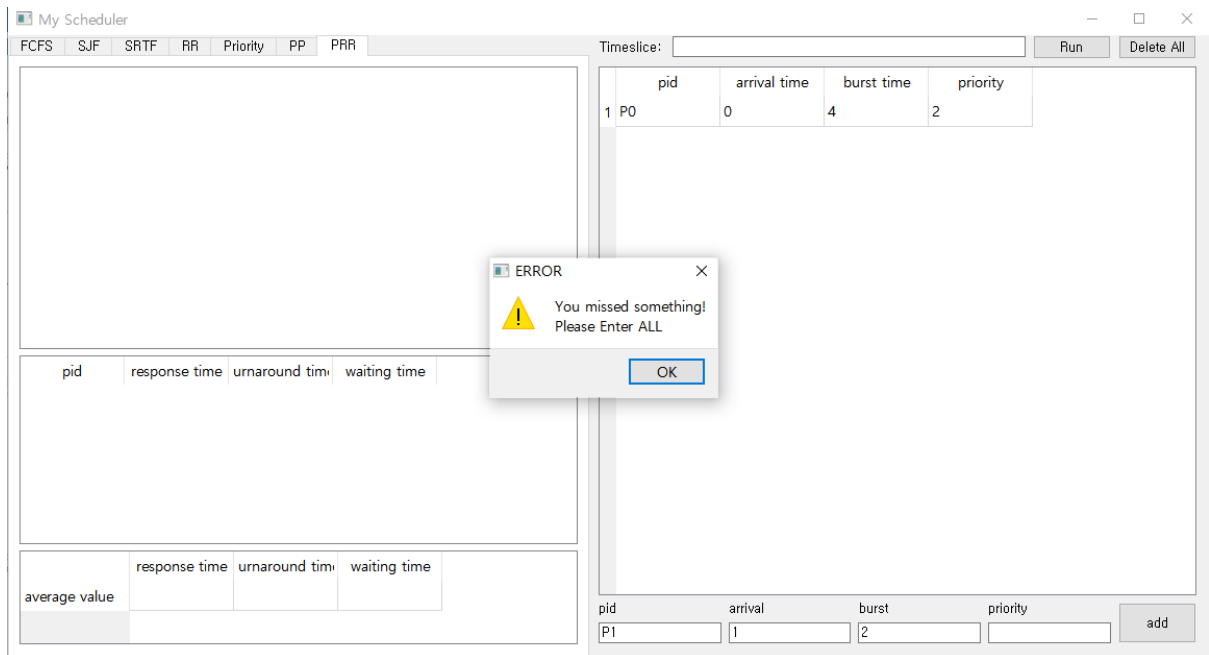
: 입력된 프로세스 테이블에 추가됨을 확인할 수 있음.

The screenshot shows the 'My Scheduler' application window after clicking the 'add' button. The process P0 has been added to the table in the bottom-right area. The table now contains one row with the following data: pid: P0, arrival time: 0, burst time: 4, priority: 2.

pid	arrival time	burst time	priority
1 P0	0	4	2

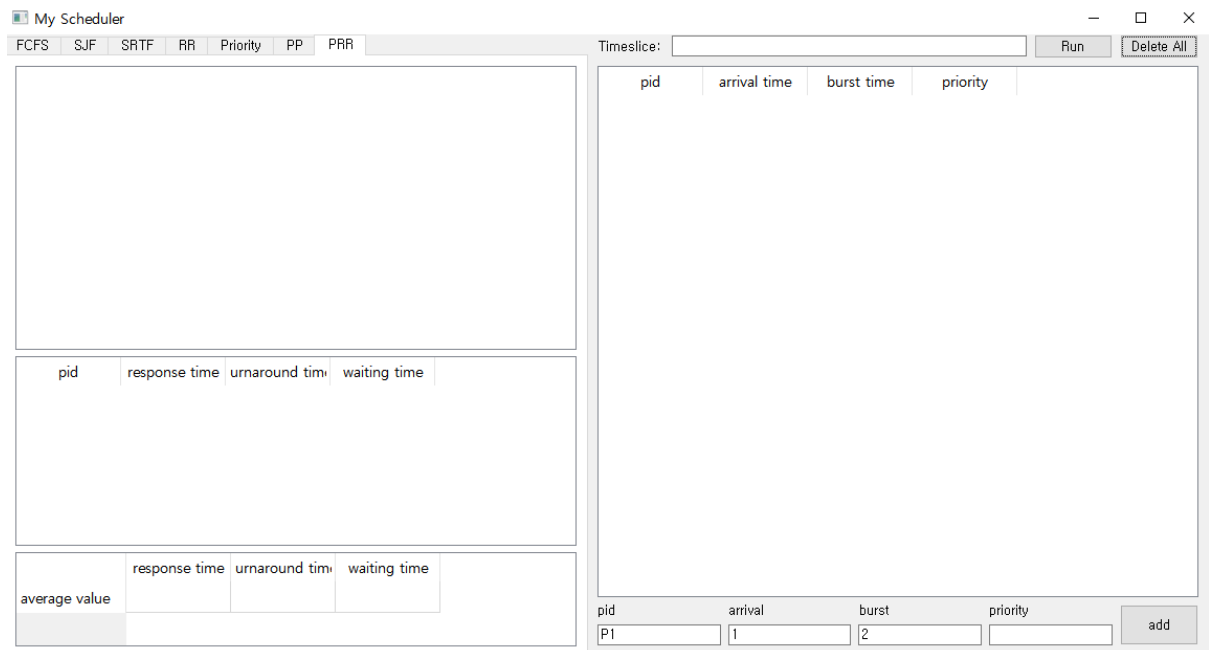
4개 입력 중 빈칸이 있을 경우 (해당 화면은 priority가 빠져 있는 경우)

: 입력이 완료되지 않았음을 에러 메시지로 발생



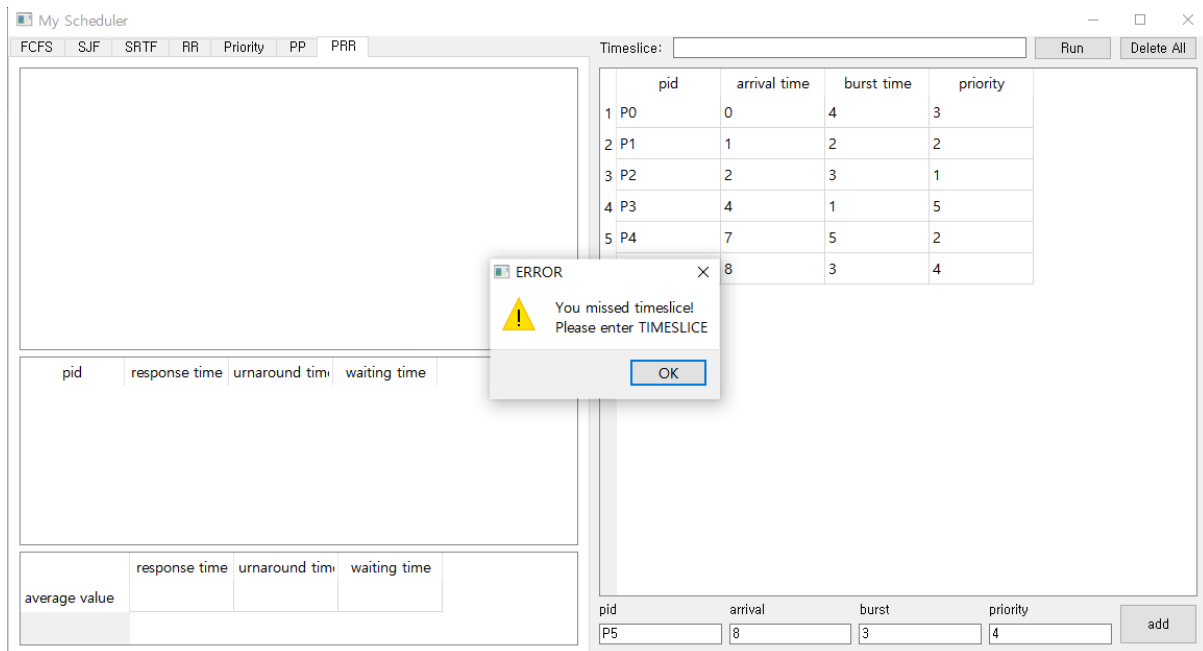
Delete All 버튼을 눌렀을 경우 : 초기화

: 입력했었던 모든 프로세스에 대한 정보를 clear



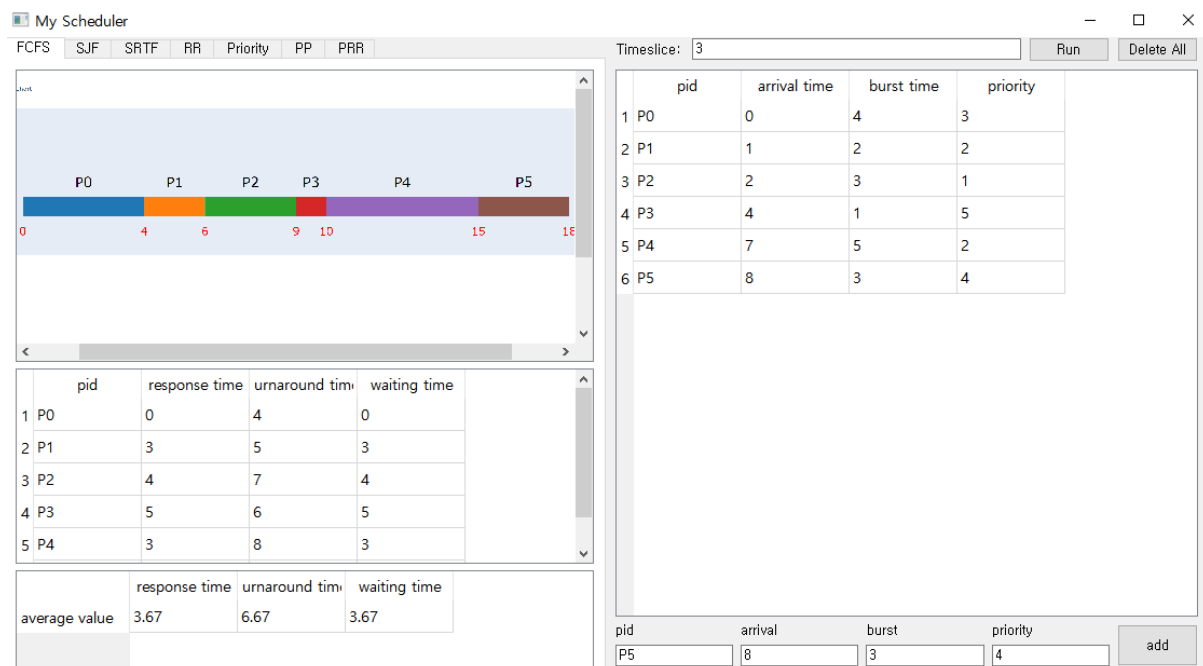
Run 버튼 클릭 : Timeslice 미입력 시

: timeslice를 입력하지 않으면 에러메시지 발생.

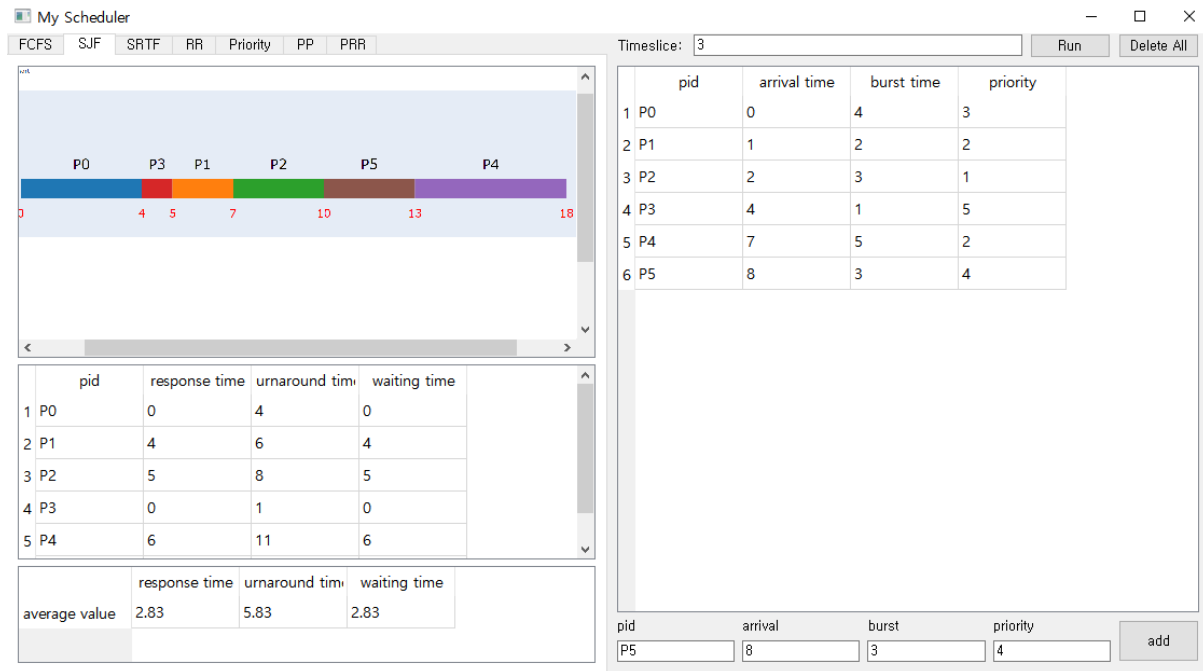


모든 프로세스 정보와 timeslice를 입력한 후 Run버튼을 클릭하면,
모든 스케줄러 실행한 후 탭별로 그 결과를 확인할 수 있다.

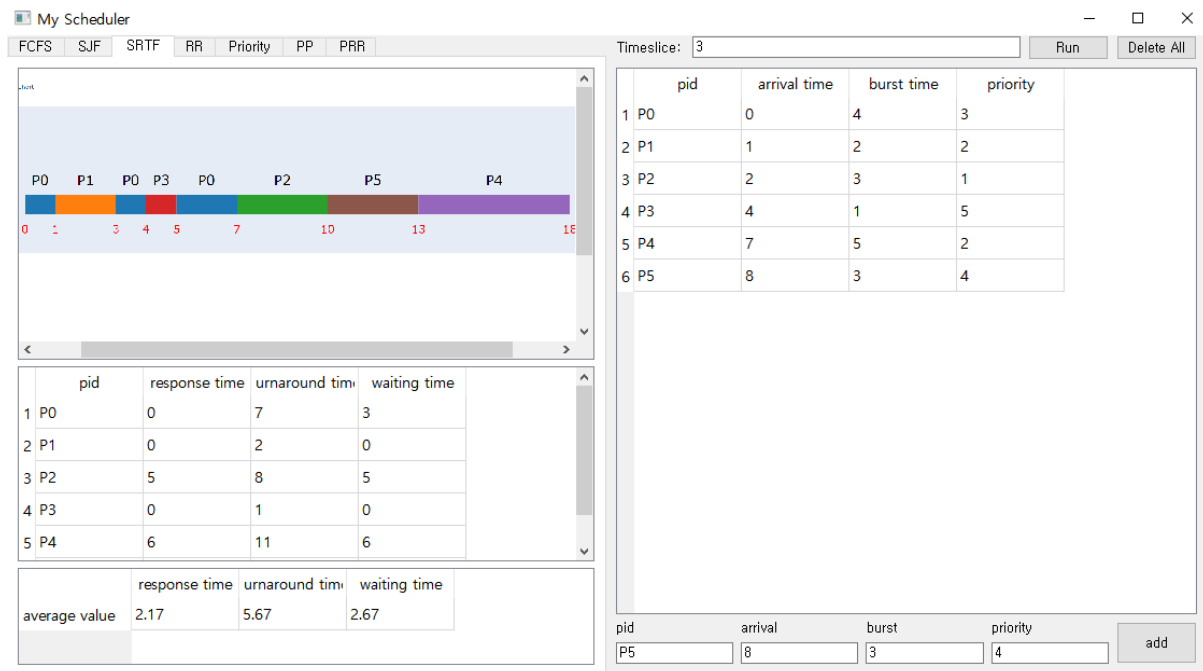
Run 클릭 시 : FCFS 결과 화면



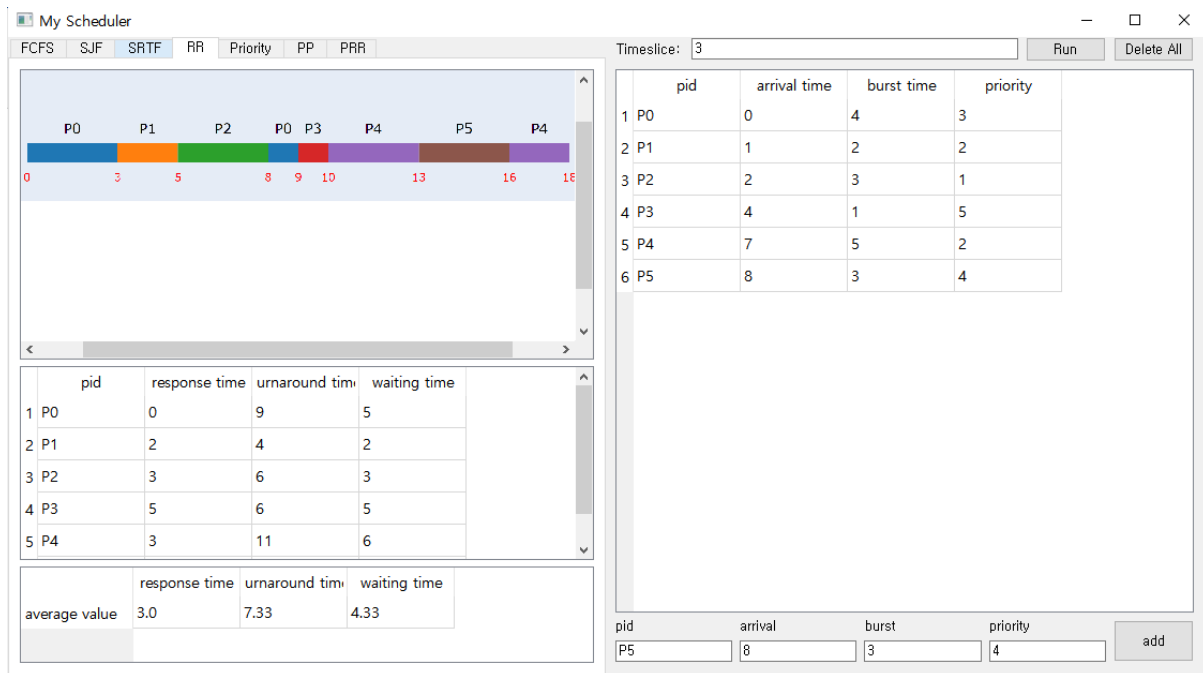
Run 클릭 시 : ShortestJobFirst 결과 화면



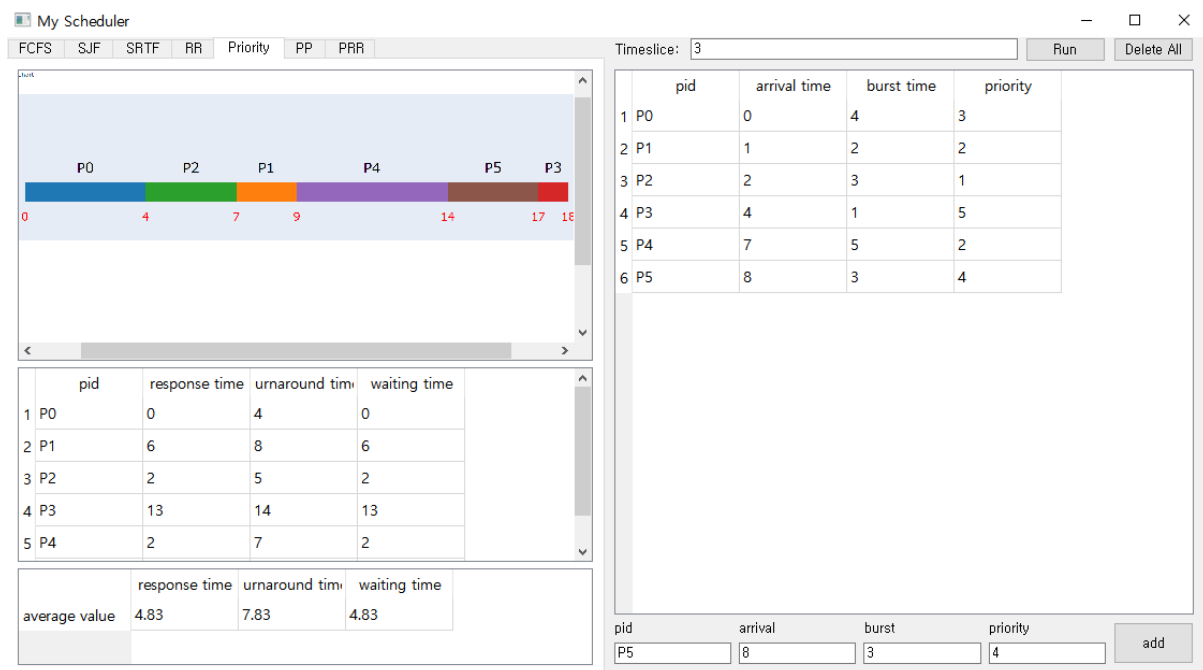
Run 클릭 시 : Shortest Remaining Time First



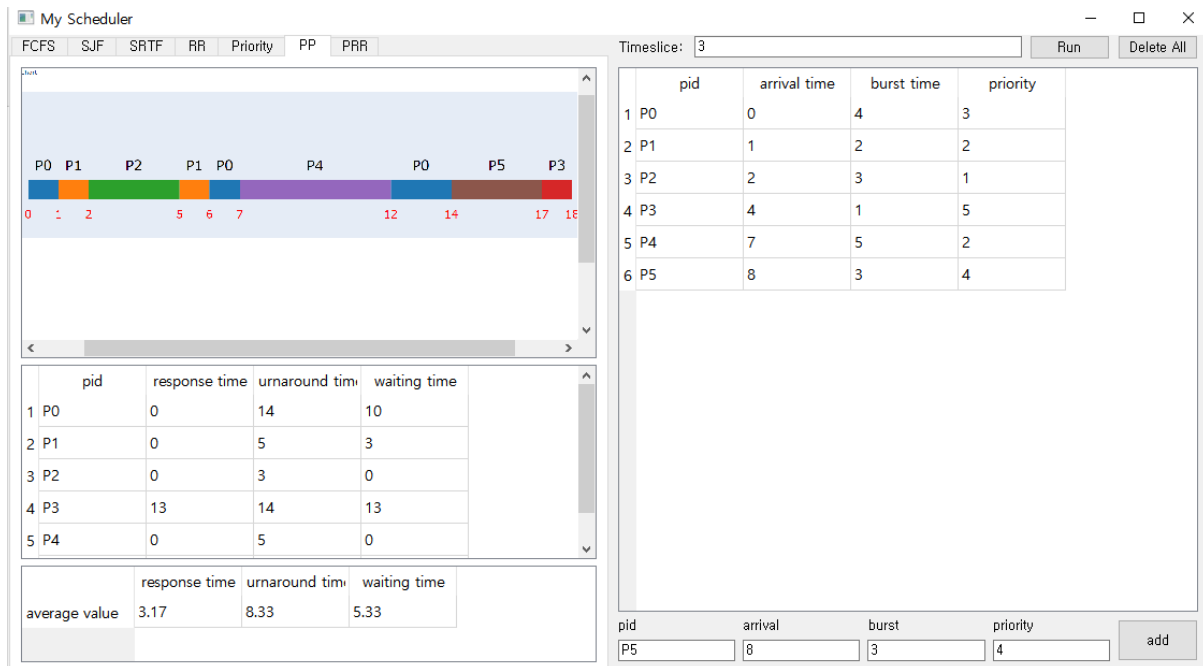
Run 클릭 시 : Round Robin



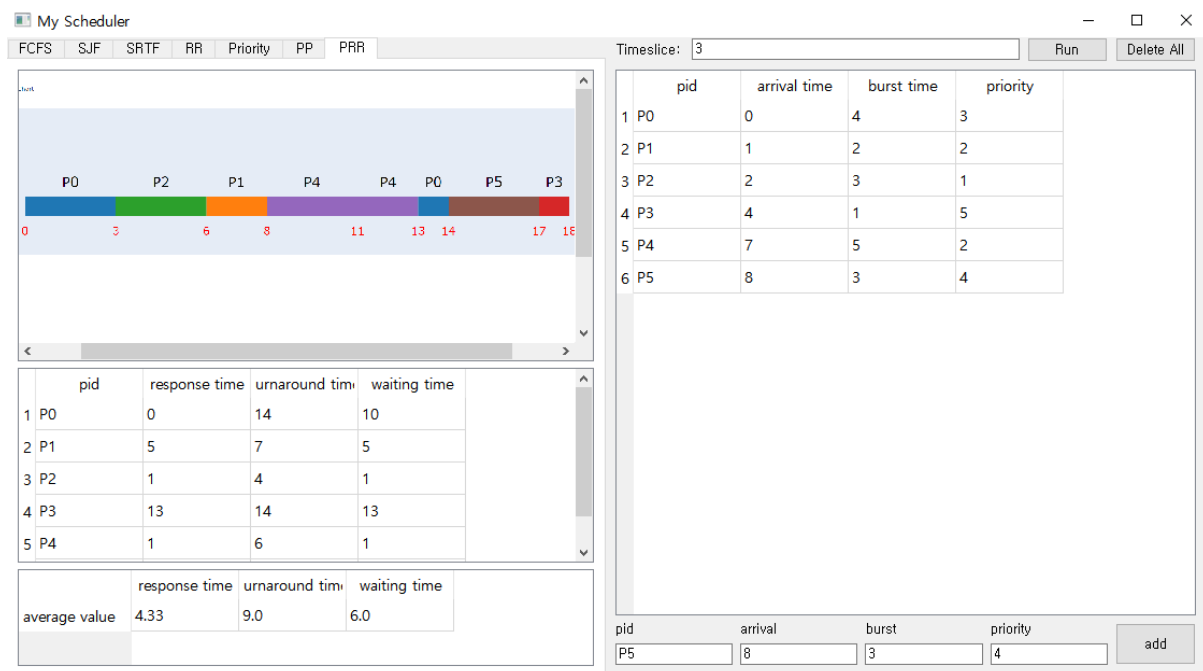
Run 클릭 시 : Priority



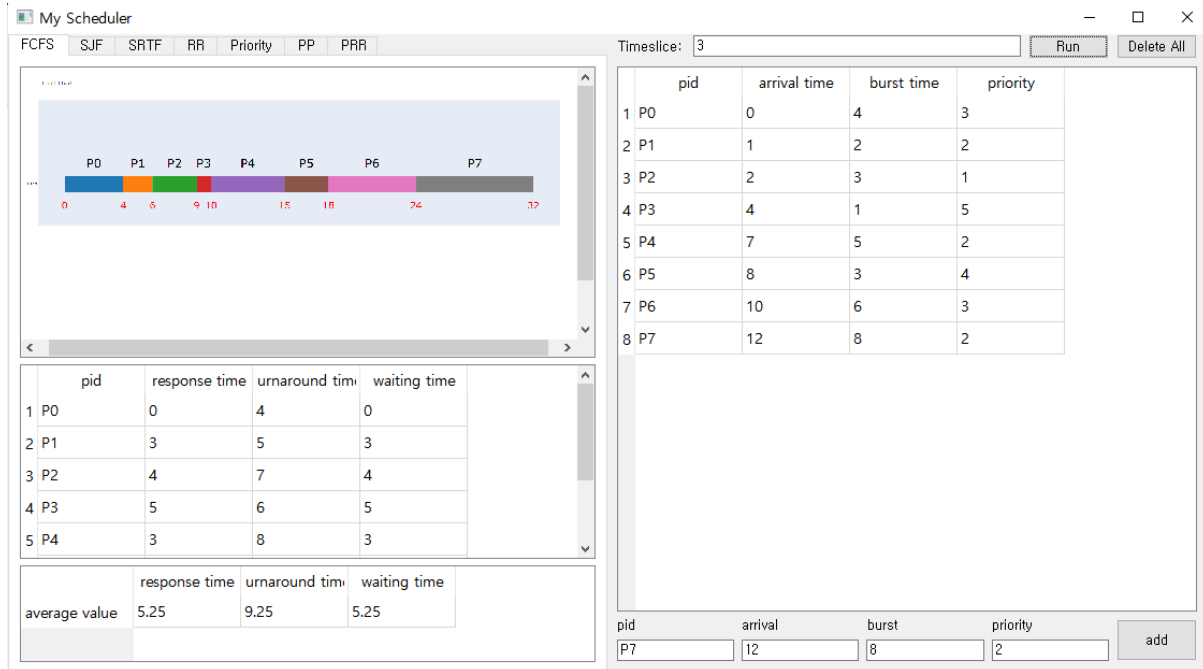
Run 클릭 시 : Preemptive Priority



Run 클릭 시 : Round Robin with Priority



새로운 프로세스 추가한 후 Run 클릭 시



Handler의 프로세스 리스트에 기존 프로세스가 이미 저장되어 있기 때문에 새 프로세스 정보 추가하더라도 바로 실행할 수 있음을 보인다. 위 사진은 FCFS의 결과이다.

V. 느낀 점

- 임정섭 – 처음으로 랜덤으로 팀을 구성하여 팀플을 진행하였다. 사람들과 알아가기부터 시작하는 것은 처음이었고, 소통을 위해서도 추가적인 노력이 필요했다. 또한, 다양한 사람들과 함께한 만큼 참신한 도구도 많이 알게 되었다. 협업을 위한 깃허브도 처음이었는데, Github conflict때문에 많이 애먹었다. 개발 과정 시 pull하는 습관이 되어 있지 않아서 file conflict가 빈번하게 발생하여 해결하는데 애를 많이 먹었다. 추상과 상속, 모듈화를 활용하며 객체지향 개념에 대해서 더 깊게 이해할 수 있었다.

- 이영섭 – notion, slack github review, branch 등 일정과 코드 관리를 위해 다양한 tool을 사용하는 방법을 배웠다.

조장님과 조원들이 일정이 지연되지 않도록 항상 잘 이끌어 주셨다. 덕분에 팀프로젝트란 이런 것이구나 느꼈고 같이 협력하는 것이 즐거웠다.

자주 회의를 하고, 의견을 교류하고 피드백을 하는 활동에서 몰랐던 부분을 알게 되고 혼자가 아니라는 생각 때문에 더 동기부여가 되었다.

클래스 다이어그램, 개발일정 설정 등 문서화 관련으로도 많은 공부를 할 수 있었다.

어떤 UI 디자인이 유저에게 편리할까 고민하는 과정을 통해 소프트웨어 개발에 큰 재미를 느꼈다.

- 박상호 – 어플 개발을 따로 해본 적이 없어서 모든 것이 새로이 배우는 수준이었기에 조장님께서 해보라는 것부터 우선 공부하였고 깃허브와 이전에는 사용하지 않았던 여러 협업도구인 notion, slack 등을 활용하여 많은 회의와 의견교류를 하여 보다 효율적인 작업 진행하는 것을 배웠다. 그래서 그나마 잘 할 수 있는 PPT 제작과 보고서 작성을 맡았다. 그저 파이썬을 할 줄 안다고 해서 뭔가 할 수 있을 줄 알았던 내게 아직은 내가 많이 부족하다는 것임을 깨달았고 앞으로 깃허브를 활용하여 다른 사용자들의 개발 과정도 살피고 어플도 제작해보며 코딩실력을 늘려가야겠다고 생각했다.
- 진시윤 – pyqt5, pyinstaller 등 다양한 패키지를 사용하는 방법을 배웠다. 개발과정 막바지에 pyinstaller를 활용하여 실행파일을 만드는데 오류가 있어 마음을 졸였다. Module 설치 시 conda install이 아닌 pip를 활용해야 한다는 점, spec file의 datas에 ui files를 추가해야 한다는 점, explicit import가 아닌 numpy와 pandas module을 spec file의 hidden_import에 추가해야 한다는 점 등을 배웠다.