

팀 프로젝트

프로세스 스케줄링 시뮬레이터

|  |  |
| --- | --- |
| 과목명 | 운영체제 |
| 분반 | 03 분반 |
| 교수 | 김덕수 교수님 |
| 팀 이름 | SO-SO |
| 팀장 | 2019136108 이해민 |
| 팀원 | 2019136047 김형구 |
| 2019136050 박금도 |
| 2019136053 박상준 |

**🛢목차**

1. 서론
2. 팀원 및 역할 소개
3. 보고서 소개
4. 프로그램 소개
5. 본론
6. 기능 소개
7. 주요 기능
8. 부가 기능
9. 클래스별 상세 설명
10. application 패키지
11. core 패키지
12. algorithm 패키지
13. 코드 동작 과정
14. 코드 시연
15. 결론
16. 개인별 소감 & 성찰

**1. 서론**

1. **팀원 및 역할 소개**

팀원들이 각자 분담된 주 역할을 관리하고 보고서를 포함한 프로젝트를 함께 처리해가며 과제 수행하였다.

|  |  |  |
| --- | --- | --- |
| 학번 | 이름 | 역할 |
| 2019136108 | 이해민 | 조장, 보고서 작성, 프로젝트 총괄 |
| 2019136047 | 김형구 | 알고리즘, UI, 오류 메시지, 도움말 구현 |
| 2019136050 | 박금도 | PPT제작 및 발표, 알고리즘, 소비전력 구현 |
| 2019136053 | 박상준 | 알고리즘, UI, 멀티코어 구현 |

1. **보고서 소개**

이 보고서는 프로세스 스케줄링 시뮬레이터 프로그램에 관한 보고서이다. 스케줄링의 기본 알고리즘 5가지(FCFS, RR, SPN, SRTN, HRRN)와 우리 팀의 독자적인 알고리즘이 동작하는 과정을 상세하게 보여준다. 각 알고리즘 별로 클래스에 정의된 변수와 메소드에 관해 설명하고 클래스 간의 관계를 나타낸다. 또한 실제 프로그램이 동작하는 화면과 결과에 관해 설명하고 마지막으로 개인별 소감과 성찰이 작성되어 있다.

1. **프로그램 개요**

이 프로그램은 프로세스의 스케줄링을 시뮬레이팅할 수 있는 프로그램이다. 기본적인 알고리즘 5가지와 자체적으로 만든 OBG(One-By-Group)라는 알고리즘을 선택하여 프로세스 스케줄링을 할 수 있다. AT와 BT에 값을 입력하고, 프로세서가 사용할 코어를 선택하고 실행하면 각 코어에서 사용하는 소비전력과 스케줄링 과정을 간트차트로 실시간으로 확인할 수 있다. 스케줄링 도중에는 다른 조작을 할 수 없도록 Run 버튼을 포함한 모든 버튼은 비활성화된다. 스케줄링이 종료되면 오른쪽에 위치한 Output Table에서 WT, TT, NTT를 확인할 수 있고 프로세스들의 평균 응답 시간을 계산하여 표시한다. 이 외에 이 프로그램을 사용하기 위해 사용자가 유의해야 할 내용을 help 탭에서 확인할 수 있다.

**2. 본론**

1. **기능 소개**
2. **주요 기능**

● 입력

1) 알고리즘 선택

- 알고리즘 선정 : select box를 이용해 스케줄링에 사용할 알고리즘을 선택한다.

- 타임 퀀텀 입력 : RR 알고리즘 선택 시 타임 퀀텀 입력란 활성화한다.

2) 코어 제어

- 코어 선택 : 스케줄링에 사용할 프로세서를 Radio Button(Off, P-Core, E-Core)를 이용하여 선택한다.

3) 프로세스 입력

- AT, BT 입력 : AT와 BT를 입력한다.

- Add 버튼 : 프로세스 이름이 Auto\_increment 되어 프로세스 리스트에 추가된다.

- Reset 버튼 : 추가된 프로세스를 초기화한다.

- 그룹 번호 입력 : OBG 알고리즘 선택 시 그룹 번호 입력란을 활성화한다.

- input Table : 입력된 프로세스의 이름과 AT, BT, GN(OBG 선택시) 을 표시한다.

● 출력

1) 실행

- Run 버튼 : 알고리즘 스케줄링을 실행한다.

2) 결과 출력

- Output Table : 스케줄링이 완료되면 프로세스의 WT, TT, NTT 등을 계산하여 표시한다.

- 소비전력 출력 : 프로세스가 사용한 소비전력을 표시한다.

- 평균 응답 시간 출력 : 스케줄링이 완료되면 평균 응답 시간을 출력한다.

- 간트차트 : 실시간으로 스케줄링되는 과정을 가시화하여 출력한다.

1. **부가 기능**

● 입력 제한 : 타임 퀀텀이나 AT, BT같은 입력란에 숫자를 제외한 문자를 입력하거나 너무 큰 숫자를 입력 받을 수 없도록 한다.

● 경고창 : 올바른 입력이 주어지지 않으면 경고창을 통해 사용자에게 알림을 준다.

● 도움말 : 입력 제한이나 경고창에 관한 내용 및 프로그램의 간단한 사용방법에 대해 설명한다.

1. **클래스별 상세 설명**

**(1) Application 패키지**

Application 패키지는 JavaFX 애플리케이션을 시작하고 실행하는 데 필요한 기본적인 구조를 제공한다. 다음과 같은 클래스와 fxml 파일로 이루어져 있다.

● Controller.java : 프로그램 기능을 전체적으로 제어

● LimitedTextField.java : 입력 길이를 제한

● Main.java : 프로그램의 시작 환경을 제공

● PrintGanttChart.java : 프로세스를 처리하는 과정을 가시화

● Process.java : 프로세스에 대한 정보를 유지

● ProcessColor.java : 프로세스 ID별로 색상을 제공

● application.css : css코드를 임포트할 수 있도록 정보를 유지

● UI.fxml : 사용자에게 제공하는 UI에 대한 정보를 유지

**Main.java**

**void start(Stage primaryStage)**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세기능** | Application GUI 설정 및 초기화 |

**static void main(String[] args)**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세기능** | Application 시작 |

**Controller.java**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및 기능** |
| int | processnumber | 1 | 프로세스 번호 자동 입력을 위한 변수 |
| Button | AddButton | - | Add 버튼 |
| ResetButton | - | Reset 버튼 |
| RunButton | - | Run 버튼 |
| HelpButton | - | Help 버튼 |
| ChoiceBox<String> | algorithmBox | - | 프로세스 선택 박스 |
| Text | Avgresponsetime | - | 평균 응답 시간 |
| core1Power | - | 1번 코어 소비전력 |
| core2Power | - | 2번 코어 소비전력 |
| core3Power | - | 3번 코어 소비전력 |
| core4Power | - | 4번 코어 소비전력 |
| TextField | atText | - | AT 입력받는 텍스트필드 |
| btText | - | BT 입력받는 텍스트필드 |
| tqText | - | tq 입력받는 텍스트필드 |
| groupText | - | groupnumber 입력받는 텍스트필드 |
| TableView<Porcess> | tableView | - | 프로세스를 입력받는 TableView |
| TableColumn<Porcess, String> | Pnames | - | 프로세스 이름이 들어가는 TableColumn |
| GroupnumberColumn | - | 그룹 번호가 들어가는 TableColumn |
| TableColumn<Process, Integer> | ATColumn | - | 입력받은 AT가 들어가는 TableColumn |
| BTColumn | - | 입력받은 BT가 들어가는 TableColumn |
| TableView<Porcess> | tableView2 | - | 실행이 끝난 프로세스가 들어가는 TableView |
| TableColumn<Process, String> | nameColumn1 | - | 실행이 끝난 프로세스 이름이 들어가는 TableColumn |
| GroupnumberColumn1 | - | 실행이 끝난 프로세스 그룹 번호가 들어가는 TableColumn |
| TableColumn<Process, Integer> | ATColumn1 | - | 실행이 끝난 프로세스의 AT가 들어가는 TableColumn |
| BTColumn1 |  | 실행이 끝난 프로세스의 BT가 들어가는 TableColumn |
| WTColumn1 | - | 실행이 끝난 프로세스 WT가 들어가는 TableColumn |
| TTColumn1 | - | 실행이 끝난 프로세스의 TT가 들어가는 TableColumn |
| TableColumn<Process, Double> | NTTColumn1 | - | 실행이 끝난 프로세스의 NTT가 들어가는 TableColumn |
| RadioButton | core1\_off, core1\_p, core1\_e | - | 1번 코어의 off, p코어, e코어 상태 |
| core2\_off, core2\_p, core2\_e | - | 2번 코어의 off, p코어, e코어 상태 |
| core3\_off, core4\_p, core3\_e | - | 3번 코어의 off, p코어, e코어 상태 |
| core4\_off, core4\_p, core4\_e | - | 4번 코어의 off, p코어, e코어 상태 |
| ScrollPane | GanttChart | - | 간트차트를 표시할 스크롤이 있는 패널 |
| Timeline | timeline | - | 정해진 시간마다 주어진 코드를 실행하기 위한 변수 |
| ObservableList<Process> | observalbeList | - | 입력받는 프로세스 리스트 |
| ObservableList<Process> | obser | - | 실행이 끝난 프로세스 리스트 |

**void initialize(URL arg0, ResourceBundle arg1)**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | JavaFX에서 ‘FXMLLoader’을 사용하여 fxml파일을 로드 할 때, 해당 컨트롤러 클래스의 객체가 생성된 후 자동으로 호출하는 초기화 메소드 |

**void showHelp()**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | 안내문의 전체적인 내용, 안내문 창의 크기, 글씨 크기 등의 정보를 저장하고 호출 시 오류 창을 출력하는 메소드 |

**boolean isInteger(String str)**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | str이 숫자인지 판별하는 메소드  str을 숫자로 변경한 후 오류가 발생하지 않을시 True, 오류가 발생시 False를 반환 |

**void warning(String str, String str1)**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | 프로그램 사용시 발생할 수 있는 오류들의 정보를 저장하고 호출 시 오류 창을 출력하는 메소드 |

**boolean isInvalidTimeQ()**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | TImeQ(타임 퀀텀)으로 인해 발생할 수 있는 오류들의 정보를 저장하고 호출시 오류 창을 출력하는 메소드 |

**Process.java**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및 기능** |
| SimpleStringProperty | processName | - | 프로세스의 이름을 저장 |
| Groupnumber | - | 프로세스의 그룹 번호 저장 |
| SimpleIntegerProperty | arrivalTime | - | 프로세스의 도착 시간 저장 |
| burstTIme | - | 프로세스의 실행 시간 저장 |
| waitingTime | - | 프로세스의 대기 시간 저장 |
| turnaroundTime | - | 프로세스의 반환 시간 |
| remainBurstTime | - | 프로세스의 남은 실행시간 |
| timeQ | - | 타임퀀텀 시간 |
| remainTimeQ | - | 남은 타임 퀀텀시간 |
|  | SimpleDoubleProperty | NormalizedTT | - | 프로세스의 정규화된 반환시간 |

**Process(String processName, int arrivalTime, int burstTime)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | processName, arrivalTime, burstTime을 넣고 객체 생성 |

**Process(String processName, int arrivalTime, int burstTime, String groupnumber)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | processName, arrivalTime, burstTime, groupnumber을 넣고 객체 생성 |

**public String getProcessName()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | ProcessName를 조회 |

**public Integer getArrivalTime()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | ArrivalTime 조회 |

**public Integer getBurstTime()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | BurstTime 조회 |

**public Integer getWaitingTime()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | WaitingTime 조회 |

**public Integer getturnaroundTime()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | turaroundTime 조회 |

**public Integer getNormalizedTT()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | NormalizedTT 조회 |

**public Integer getremainburstTime()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | remainburstTime 조회 |

**public Integer getTimeQ()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | TimeQ 조회 |

**public Integer getremainTimeQ()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | remainTimeQ 조회 |

**public String getgroupnumber()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | groupnumber 조회 |

**public void setProcessName(String processName)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | ProcessName 값을 변경 |

**public void setArrivalTime(Integer arrivalTime)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | ArrivalTime 값을 변경 |

**public void setBurstTime(Integer burstTime)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | BurstTime 값을 변경 |

**public void setWaitingTime(Integer waitingtimes)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | WaitingTime 값을 변경 |

**public void setNormalizedTT(double d)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | NormalizedTT 값을 변경 |

**public void setturnaroundTime(Integer turnaroundTime)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | turnaroundTime 값을 변경 |

**public void setremainburstTime(Integer time)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | remainburstTime 값을 변경 |

**public void setTimeQ(Integer time)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | TimeQ 값을 변경 |

**public void setremainTimeQ(Integer time)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | remainTimeQ 값을 변경 |

**public void setGroupnumber(String groupnumber)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | groupnumber 값을 변경 |

**public SimpleStringProperty processNameProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | processName 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleIntegerProperty arrivalTimeProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | ArrivalTime 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleIntegerProperty burstTimeProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | burstTime 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleIntegerProperty waitingTimeProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | waitingTime 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleIntegerProperty turnaroundTimeProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | turnaroundTime 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleDoubleProperty NormalizedTTProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | NormalizedTT 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleIntegerProperty remainBurstTimeroperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | remainBurstTIme 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleIntegerProperty TimeQProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | TimeQ 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleIntegerProperty remainTimeQProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | remainTimeQ 값을 테이블에 띄울 수 있게 속성 변경 |

**public SimpleStringProperty groupnumberProperty()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | groupnumber 값을 테이블에 띄울 수 있게 속성 변경 |

**LimitedTextField.java**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및 기능** |
| static final int | MAX-LENGTH | 5 | textField에 입력할 수 있는 길이 |

**LimitedTextField()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | textField에 길이가 MAX\_LENGTH보다 길거나 같으면 더이상 적히지 않게 하는 메소드 |

**PrintGhanttchart.java**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및 기능** |
| VBox | v | new VBox | 행 수, box들을 행별로 붙여서 저장 |
| ProcessColor | processcoler | new ProcessColor() | 간트차트 출력을 위한 프로세스 색 |

**public void print(Core[] Corelist, ScrollPane GanttChart, HBox hBox, int time)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 스크롤 되는 패널에 간트차트를 출력  ① 프로세스 이름에서 숫자만 가져온 후 1을 뺌(인덱스 접근)  ② 코어에 프로세스가 들어 왔을 경우 ①에서 구한 숫자 별로 색상 지정  ③ 코어에 프로세스가 들어 오지 않았다면 빈칸으로 표시  ④ 스크롤바 갱신 |

**ProcessColor.java**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및 기능** |
| String[] | color | new String[15] | 15개 용량의 String 리스트 생성 |

**public ProcessColor()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 프로세스별로 색상 지정 |

**public String getColor()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 프로세스의 색상 조회 |

**(2) Core 패키지**

Core 패키지는 코어마다 공통적으로 사용하는 필드 변수나, 메소드 혹은 P코어와 E코어에서 다르게 사용되는 필드 변수나 메소드들이 구현되어 있습니다.

● Core.java : Ecore와 Pcore가 공통적으로 사용하는 변수와 메소드 제공

● Ecore.java : E코어에서만 사용되는 전력량, 실행 시간등 정보 저장

● Pcore.java : P코어에서만 사용되는 전력량, 실행 시간등 정보 저장

**Core.java**

**public class Core**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및 기능** |
| Process | process | - | 프로세스 정보 저장 |
| boolean | visited | false | 프로세스의 방문 여부 |
| boolean | prevUsed | false | 코어가 쉬고 있는지 여부 |
| double | powerConsumption | 0 | 소비전력 |

**public Process getProcess()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 프로세스 조회 |

**public void setProcess(Process process)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 프로세스 값을 변경 |

**public boolean isVisited()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 프로세스가 방문했는지 확인 |

**public void setVisited(boolean visited)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 프로세스가 방문한 여부에 대한 값을 변경 |

**public boolean isprevUsed()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 프로세스가 이전에 쉬고 있엇는지 확인 |

**public void setprevUsed(boolean prevUsed)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | 프로세스가 이전에 쉬고 있었는지에 대한 값을 변경 |

**public double getPowerConsumption()**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | PowerConsumption 값을 조회 |

**public void setPowerConsumption(double powerConsumption)**

|  |  |  |
| --- | --- | --- |
| 메소드 | 상세 기능 | PowerConsumption 값을 변경 |

Pcore.java와 Ecore.java가 같은 값을 가지는 메소드들만 여기서 정의하고 각각의 Core에서 다르게 작동되는 메소드들은 각각의 클래스에서 정의 되어있습니다.

**public void setComponent(int time) {} @Override**

**public void setComponent\_p(int time) @Override**

**public void emptyCore() {} @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | Core에 프로세스 사용 여부를 빈칸으로 할당  ① Core의 Process변수에 null 할당  ② Core의 visited변수에 false 할당 |

**public void setPower() @Override**

**public boolean work(int time) @Override**

**public boolean RRwork(int time, LinkedList<Process> waitingList) @Override**

**public boolean SRTNwork(int time, LinkedList<Process> waitingList) @Override**

**Ecore.java**

**public class Ecore extends Core**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및 기능** |
| static final int | excuteTime | 1 | 1초에 처리하는 일의 양 |

**public void setComponent(int time) {} @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | TurnaroundTime, NormallizedTurnaroundTime, WaitingTime 계산하여 설정  ① time(현재시간) – ArrivalTime + 1을 한 값을 TurnaroundTime에 저장  ② TurnaroundTime을 BurstTime으로 나눈 값을 NormallizedTurnaroundTime에 저장 (Math.round함수를 통해 소수 2째자리 까지만 저장)  ③ TurnaroundTime에서 BurstTime을 뺀 값을 WaitingTime에 저장 |

**public void setComponent\_p(int time) {} @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | TurnaroundTime, NormallizedTurnaroundTime, WaitingTime 계산하여 설정(Preemptive(선점) 스케줄링 알고리즘을 위한 메소드)  ① time(현재시간) – ArrivalTime + 1을 한 값을 TurnaroundTime에 저장  ② TurnaroundTime을 BurstTime으로 나눈 값을 NormallizedTurnaroundTime에 저장 (Math.round함수를 통해 소수 2째자리 까지만 저장) |

**public void setPower() @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | 코어에서 소비되는 전력을 갱신하는 메소드  ① 이전에 코어가 쉬고 있었다면 시동전력 0.1 추가  ② prevUsed 값을 Ture로 변경  ③ 코어가 사용되면 1초마다 전력 1 추가 |

**public boolean work(int time) @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | 비선점 스케줄링 알고리즘이 공통적으로 사용하는 메소드  ① setPower()를 실행하여 소비전력을 갱신  ② remainBursrtTime을 excuteTime만큼 감소  ③ remainBursrtTime이 0보다 작으면(할당된 BurstTime만큼의 일을 다 끝낸 경우) setComponent(), emptyCore() 실행하고 ture 반환  ④ remainBursrtTime이 0보다 크면(처리해야하는 작업이 남은 경우) false 반환 |

**public boolean RRwork(int time, LinkedList<Process> waitingList) @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | TimeQ초에 excute Time(1)의 일을 하고, 일이 남았을 경우waitingList에 추가하고 False를 반환  ① setPower()를 실행하여 소비전력을 갱신  ② remainBursrtTime을 excuteTime만큼 감소  ③ 남은 timeQ 1초 감소  ④ reaminTimeQ가 0이고 remainBurstTime 이 0 보다 작다면(timeQ만큼의 일을 다하고 BurstTIme만큼의 일을 다 끝낸 경우) setComponent\_(), emptyCore()를 실행하고 ture 반환하고 remainTimeQ는 0이지만 remainBurstTime이 0보다 클 경우(TimeQ의 일은 다했지만 BurstTime 만큼의 일을 다 하지 않았을 경우) emptyCore()를 실행하고 false를 반환  ⑤ BurstTime만큼의 일을 다 했을 경우 setComponent\_p(), emptyCore()를 실행하고 false를 반환 |

**public boolean SRTNwork(int time, LinkedList<Process> waitingList) @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | 1초에 excute Time(1)일을 하고, 일이 남았을 경우 waitingList에 추가하고 False를 반환  ① setPower()를 실행하여 소비전력을 갱신  ② remainBursrtTime을 excuteTime만큼 감소  ③ remainBurstTime이 0보다 작거나 같다면(BurstTime만큼의 일을 다 한 경우) setComponent\_p(), emptyCore()를 실행하고 true를 반환  remainBurstTIme이 0보다 크다면(BurstTime만큼의 일을 다 하지 한 경우) waitingList에 추가, emptyCore()를 실행하고 false를 반환 |

**Pcore.java**

**public class Pcore**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및 기능** |
| static final int | excuteTime | 2 | 1초에 처리하는 일의 양 |

**public void setComponent(int time) {} @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | TurnaroundTime, NormallizedTurnaroundTime, WaitingTime 계산하여 설정  ① time(현재시간) – ArrivalTime + 1의 값을 TurnaroundTime에 저장  ② TurnaroundTime을 BurstTime으로 나눈 값을 NormallizedTurnaroundTime에 저장 (Math.round함수를 통해 소수 2째자리 까지만 저장)  ③ BurstTime이 짝수라면 (turnaroundTime – BurstTime) / 2의 값을 WaitingTime에 저장  ④ BurstTime이 홀수라면 (turnaroundTime – BurstTime + 1) / 2의 값을 WaitingTime에 저장 |

**public void setComponent\_p(int time) {} @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | TurnaroundTime, NormallizedTurnaroundTime, WaitingTime 계산하여 설정(Preemptive(선점) 스케줄링 알고리즘을 위한 메소드)  ① time(현재시간) – ArrivalTime + 1을 한 값을 TurnaroundTime에 저장  ② TurnaroundTime을 BurstTime으로 나눈 값을 NormallizedTurnaroundTime에 저장 (Math.round함수를 통해 소수 2째자리 까지만 저장) |

**public void setPower() @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | 코어에서 소비되는 전력을 갱신하는 메소드  ① 이전에 코어가 쉬고 있었다면 시동전력 0.5 추가  ② prevUsed 값을 Ture로 변경  ③ 코어가 사용되면 1초마다 전력 3 추가 |

**public boolean work(int time) @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | 비선점 스케줄링 알고리즘이 공통적으로 사용하는 메소드  ① setPower()를 실행하여 소비전력을 갱신  ② remainBursrtTime을 excuteTime만큼 감소  ③ remainBursrtTime이 0보다 작으면(할당된 BurstTime만큼의 일을 다 끝낸 경우) setComponent(), emptyCore() 실행하고 ture 반환  ④ remainBursrtTime이 0보다 크면(처리해야하는 작업이 남은 경우) false 반환 |

**public boolean RRwork(int time, LinkedList<Process> waitingList) @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | TimeQ초에 excute Time(1)의 일을 하고, 일이 남았을 경우waitingList에 추가하고 False를 반환  ① setPower()를 실행하여 소비전력을 갱신  ② remainBursrtTime을 excuteTime만큼 감소  ③ 남은 timeQ 1초 감소  ④ reaminTimeQ가 0이고 remainBurstTime 이 0 보다 작다면(timeQ만큼의 일을 다하고 BurstTIme만큼의 일을 다 끝낸 경우) setComponent\_(), emptyCore()를 실행하고 ture 반환하고 remainTimeQ는 0이지만 remainBurstTime이 0보다 클 경우(TimeQ의 일은 다했지만 BurstTime 만큼의 일을 다 하지 않았을 경우) emptyCore()를 실행하고 false를 반환  ⑤ BurstTime만큼의 일을 다 했을 경우 setComponent\_p(), emptyCore()를 실행하고 false를 반환 |

**public boolean SRTNwork(int time, LinkedList<Process> waitingList) @Override**

|  |  |  |
| --- | --- | --- |
| **메소드** | **상세 기능** | 1초에 excute Time(1)일을 하고, 일이 남았을 경우 waitingList에 추가하고 False를 반환  ① setPower()를 실행하여 소비전력을 갱신  ② remainBursrtTime을 excuteTime만큼 감소  ③ remainBurstTime이 0보다 작거나 같다면(BurstTime만큼의 일을 다 한 경우) setComponent\_p(), emptyCore()를 실행하고 true를 반환  remainBurstTIme이 0보다 크다면(BurstTime만큼의 일을 다 하지 한 경우) waitingList에 추가, emptyCore()를 실행하고 false를 반환 |

**(3) Algorithm 패키지**

Algorithm 패키지에는 비선점 스케줄링 알고리즘(FCFS, SPN, HRRN, OBG) 4가지와 선점 스케줄링 알고리즘(RR, SRTN) 2가지으로 구성 되어있습니다. 비선점 스케줄링 알고리즘은 이미 사용되는 코어를 뺏지(선점)하지 못하고 사용이 끝날 때까지 기다려야 하는 방식이고, 선점 스케줄링 알고리즘은 코어를 조건에 따라 뺏어(선점)할 수 있는 알고리즘입니다. 각각의 알고리즘에 대한 구체적인 설명은 알고리즘 필드와 메소드를 설명하면서 이어가도록 하겠습니다. 6가지 알고리즘 중에서 사용자가 필요로 하는 알고리즘을 호출하여 사용하는 방식으로 구현되어 있습니다.

**FCFS.java**

FCFS(First Come First Service)는 도착시간이 가장 빠른 프로세스를 먼저 코어를 할당해주는 기법이다. **public void run(Core[] coreList, LinkedList<Process> waitingList, LinkedList<Process> endList, ObservableList<Process> obser, ObservableList<Process> observableList, ScrollPane GanttChart, HBox hBox, Text[] corePower, Text Avgresponsetime, TableView<Process> tableView2, Button[] Buttons, ChoiceBox<String> choicebox)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필드** | **Data Type** | **Variable Name** | **Value** | **용도 및기능** |
| Core[] | coreList | - | Pcore, Ecore, off 를 받아오는 용도 |
| LinkedList<Process> | waitingList | - | 일을 하기 위해 기다리는 프로세스를 저장하는 용도 |
| LinkedList<Process> | endList | - | 일이 끝난 프로세스를 저장하는 용도 |
| ObservableList<Process> | obser | - | 입력받은 프로세스 리스트 |
| ObservableList<Process> | observableList | - | 패널에 출력을 위한 리스트 |
| ScrollPane | GanttChart | - | 스크롤이있는 패널. 간트차트를 출력하는 용도 |
| Text[] | corePower | - | 코어가 소비한 전력 출력하는 용도 |
| Text | Avgresponsetime | - | 평균 응답시간을 출력하는 용도 |
| TableView<Process> | tableView2 | - | 결과 출력하는 용도 |
| Button[] | Buttons | - | 실행 중 버튼을 잠그는 용도 |
| ChoiceBox<String> | choicebox | - | 실행 중 알고리즘 선택박스를 잠그는 용도 |
| **메소드** | **상세 기능** | ① 입력받은 프로세스의 AT가 현재시간과 같으면 waitingList에 삽입  ② 남은 코어 자리에 waitingList에 있는 프로세스들을 번호 순서대로 할당  ③ 간트차트 출력  ④ 코어별로 1초의 일을하고, BT만큼의 일을 Work메소드를 통해 다하면 endList에 삽입  ⑤ endList와 observableList가 같으면 종료 | | |

**RR.java**

RR(Round Robin)은 도착시간이 빠른 프로세스가 먼저 들어가지만 시간 제한(Time Quantum)이 존재하여 Time Quantum의 일을 하고 난 후 코어를 빠져나와 대기 큐에 대기 한 후 다시 코어가 비게 된다면 WaitingQ에 먼저 들어온 순서대로 다시 코어에 할당되는 것이 반복되는 기법이다.

**public void run(Core[] coreList, LinkedList<Process> waitingList, LinkedList<Process> endList, ObservableList<Process> obser, ObservableList<Process> observableList, ScrollPane GanttChart, HBox hBox, TextField textField4, Text[] corePower, Text Avgresponsetime, TableView<Process> tableView2,Button[] Buttons, ChoiceBox<String> choicebox)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필**  **드** | **Data Type** | **Variable Name** | **Value** | **용도 및기능** |
| Core[] | coreList | - | Pcore, Ecore, off 를 받아오는 용도 |
| LinkedList<Process> | waitingList | - | 일을 하기 위해 기다리는 프로세스를 저장하는 용도 |
| LinkedList<Process> | endList | - | 일이 끝난 프로세스를 저장하는 용도 |
| ObservableList<Process> | obser | - | 입력받은 프로세스 리스트 |
| ObservableList<Process> | observableList | - | 패널에 출력을 위한 리스트 |
| ScrollPane | GanttChart | - | 스크롤이있는 패널. 간트차트를 출력하는 용도 |
| TextField | textField4 | textField4.getText() | String 형으로 입력 받았던 TimeQuantum을 받아오는 용도. |
| Text[] | corePower | - | 코어가 소비한 전력 출력하는 용도 |
| Text | Avgresponsetime | - | 평균 응답시간을 출력하는 용도 |
| TableView<Process> | tableView2 | - | 결과 출력하는 용도 |
| Button[] | Buttons | - | 실행 중 버튼을 잠그는 용도 |
| ChoiceBox<String> | choicebox | - | 실행 중 알고리즘 선택박스를 잠그는 용도 |
| **메소드** | **상세 기능** | ① 입력받은 프로세스의 AT가 현재시간과 같으면 waitingList에 삽입  ② 남은 코어 자리에 waitingList에 있는 프로세스들을 번호 순서대로 할당  ③ 간트차트 출력  ④ 코어별로 1초의 일을하고, BT만큼의 일을 RRWork메소드를 통해 다하면 endList에 삽입  ⑤ endList와 observableList가 같으면 종료  ⑥ 결과 테이블, avgresponseTime 출력 | | |

**SPN.java**

SPN(Shortest Process Next)은 도착시간이 빠른 순서대로 코어를 할당해주지만, FCFS와 다르게 같은 시간에 들어오거나 여러 프로세스가 대기중이라면 실행시간이 가장 적은 프로세스에게 코어를 먼저 할당해주는 기법이다.

**public void run(Core[] coreList, LinkedList<Process> waitingList, LinkedList<Process> endList, ObservableList<Process> obser, ObservableList<Process> observableList, ScrollPane GanttChart, HBox hBox, Text[] corePower, Text Avgresponsetime, TableView<Process> tableView2,Button[] Buttons,ChoiceBox<String> choicebox)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필**  **드** | **Data Type** | **Variable Name** | **Value** | **용도 및기능** |
| Core[] | coreList | - | Pcore, Ecore, off 를 받아오는 용도 |
| LinkedList<Process> | waitingList | - | 일을 하기 위해 기다리는 프로세스를 저장하는 용도 |
| LinkedList<Process> | endList | - | 일이 끝난 프로세스를 저장하는 용도 |
| ObservableList<Process> | obser | - | 입력받은 프로세스 리스트 |
| ObservableList<Process> | observableList | - | 패널에 출력을 위한 리스트 |
| ScrollPane | GanttChart | - | 스크롤이있는 패널. 간트차트를 출력하는 용도 |
| Text[] | corePower | - | 코어가 소비한 전력 출력하는 용도 |
| Text | Avgresponsetime | - | 평균 응답시간을 출력하는 용도 |
| TableView<Process> | tableView2 | - | 결과 출력하는 용도 |
| Button[] | Buttons | - | 실행 중 버튼을 잠그는 용도 |
| ChoiceBox<String> | choicebox | - | 실행 중 알고리즘 선택박스를 잠그는 용도 |
| **메소드** | **상세 기능** | ① 입력받은 프로세스의 AT가 현재시간과 같으면 waitingList에 삽입  ② 남은 코어 자리에 waitingList에 있는 프로세스들 중 BurstTIme이 낮은 프로세스의 인덱스를 찾고 코어에 추가 한 후 waitingList에서 삭제(코어에 빈자리가 생길때 마다 반복)  ③ 간트차트 출력  ④ 코어별로 1초의 일을하고, BT만큼의 일을 Work 메소드를 통해 다하면 endList에 삽입 생길때 마다 반복)  ⑤ endList와 observableList가 같으면 종료  ⑥ 결과 테이블, avgresponseTime 출력 | | |

**SRTN.java**

SRTN(Shortest Remaining Time Next)는 SPN과 비슷하지만 SPN과 달리 선점 스케줄링 방식으로 매 초마다 실행시간이 가장 적은 프로세스에게 코어를 선점하여 먼저 할당해주는 기법이다.

**public void run(Core[] coreList, LinkedList<Process> waitingList, LinkedList<Process> endList, ObservableList<Process> obser, ObservableList<Process> observableList, ScrollPane GanttChart, HBox hBox, Text[] corePower, Text Avgresponsetime, TableView<Process> tableView2,Button[] Buttons,ChoiceBox<String> choicebox)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필**  **드** | **Data Type** | **Variable Name** | **Value** | **용도 및기능** |
| Core[] | coreList | - | Pcore, Ecore, off 를 받아오는 용도 |
| LinkedList<Process> | waitingList | - | 일을 하기 위해 기다리는 프로세스를 저장하는 용도 |
| LinkedList<Process> | endList | - | 일이 끝난 프로세스를 저장하는 용도 |
| ObservableList<Process> | obser | - | 입력받은 프로세스 리스트 |
| ObservableList<Process> | observableList | - | 패널에 출력을 위한 리스트 |
| ScrollPane | GanttChart | - | 스크롤이있는 패널. 간트차트를 출력하는 용도 |
| Text[] | corePower | - | 코어가 소비한 전력 출력하는 용도 |
| Text | Avgresponsetime | - | 평균 응답시간을 출력하는 용도 |
| TableView<Process> | tableView2 | - | 결과 출력하는 용도 |
| Button[] | Buttons | - | 실행 중 버튼을 잠그는 용도 |
| ChoiceBox<String> | choicebox | - | 실행 중 알고리즘 선택박스를 잠그는 용도 |
| **메소드** | **상세 기능** | ① 입력받은 프로세스의 AT가 현재시간과 같으면 waitingList에 삽입  ② 남은 코어 자리에 waitingList에 있는 프로세스들 중 BurstTIme이 낮은 프로세스의 인덱스를 찾고 코어에 추가 한 후 waitingList에서 삭제(매 초마다 반복)  ③ 간트차트 출력  ④ 코어별로 1초의 일을하고, BT만큼의 일을 SRTNWork메소드를 통해 다하면 endList에 삽입  ⑤ endList와 observableList가 같으면 종료  ⑥ 결과 테이블, avgresponseTime 출력 | | |

**HRRN.java**

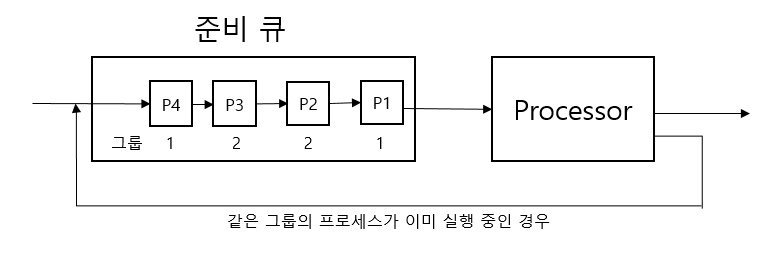
HRRN(High-Response-Ratio-Next)은은 SPN과 비슷한 기법이다. SPN에 Response ratio를 부여하여 오래 기다린 프로세스에게 먼저 코어를 할당해준다. Response ratio는 (WT + BT) / BT로 구하며 실행시간과 대기시간을 이용하여 코어를 할당해줄 프로세스를 구하는 기법이다.

**public void run(Core[] coreList, LinkedList<Process> waitingList, LinkedList<Process> endList, ObservableList<Process> obser, ObservableList<Process> observableList, ScrollPane GanttChart, HBox hBox,**

**Text[] corePower, Text Avgresponsetime, TableView<Process> tableView2,Button[] Buttons,ChoiceBox<String> choicebox)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필**  **드** | **Data Type** | **Variable Name** | **Value** | **용도 및기능** |
| Core[] | coreList | - | Pcore, Ecore, off 를 받아오는 용도 |
| LinkedList<Process> | waitingList | - | 일을 하기 위해 기다리는 프로세스를 저장하는 용도 |
| LinkedList<Process> | endList | - | 일이 끝난 프로세스를 저장하는 용도 |
| ObservableList<Process> | obser | - | 입력받은 프로세스 리스트 |
| ObservableList<Process> | observableList | - | 패널에 출력을 위한 리스트 |
| ScrollPane | GanttChart | - | 스크롤이있는 패널. 간트차트를 출력하는 용도 |
| Text[] | corePower | - | 코어가 소비한 전력 출력하는 용도 |
| Text | Avgresponsetime | - | 평균 응답시간을 출력하는 용도 |
| TableView<Process> | tableView2 | - | 결과 출력하는 용도 |
| Button[] | Buttons | - | 실행 중 버튼을 잠그는 용도 |
| ChoiceBox<String> | choicebox | - | 실행 중 알고리즘 선택박스를 잠그는 용도 |
| **메소드** | **상세 기능** | ① 입력받은 프로세스의 AT가 현재시간과 같으면 waitingList에 삽입  ② 남은 코어 자리에 waitingList에 있는 프로세스들 중 responseRatio값이 높은 프로세스의 인덱스를 찾고 코어에 추가 한 후 waitingList에서 삭제(코어에 빈자리가 생길때 마다 반복)  ③ 간트차트 출력  ④ 코어별로 1초의 일을하고, BT만큼의 일을 Work 메소드를 통해 다하면 endList에 삽입  ⑤ endList와 observableList가 같으면 종료  ⑥ 결과 테이블, avgresponseTime 출력 | | |

**자체 알고리즘 OBG (One-By-Group Algorithm)**

****

**구현 목적**

컴퓨터를 사용하면서 동시에 실행되는 경우 문제를 발생하는 프로그램들이 있다. 그런 프로그램이 동시 실행되지 않도록 프로세스를 하나의 그룹으로 묶어서 실행을 통제하기 위해 이렇게 One-By-Group Algorithm을 구상했다.

**특징**

* Non preemptive scheduling(비선점 스케줄링)
* 스케줄링 기준

• 도착 시간 (ready queue 기준)

• 먼저 도착한 프로세스를 먼저 처리

**예상 장/단점**

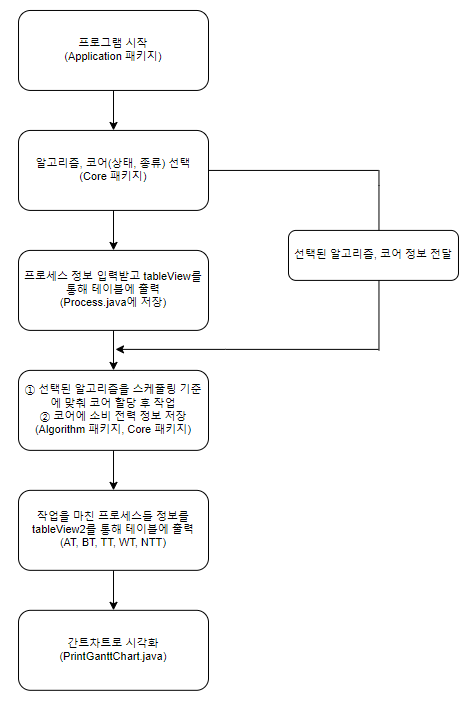
* 장점 : 프로세스 간의 충돌 방지
* 단점 : Convoy Effect, 긴 평균 응답시간

**OBG.java**

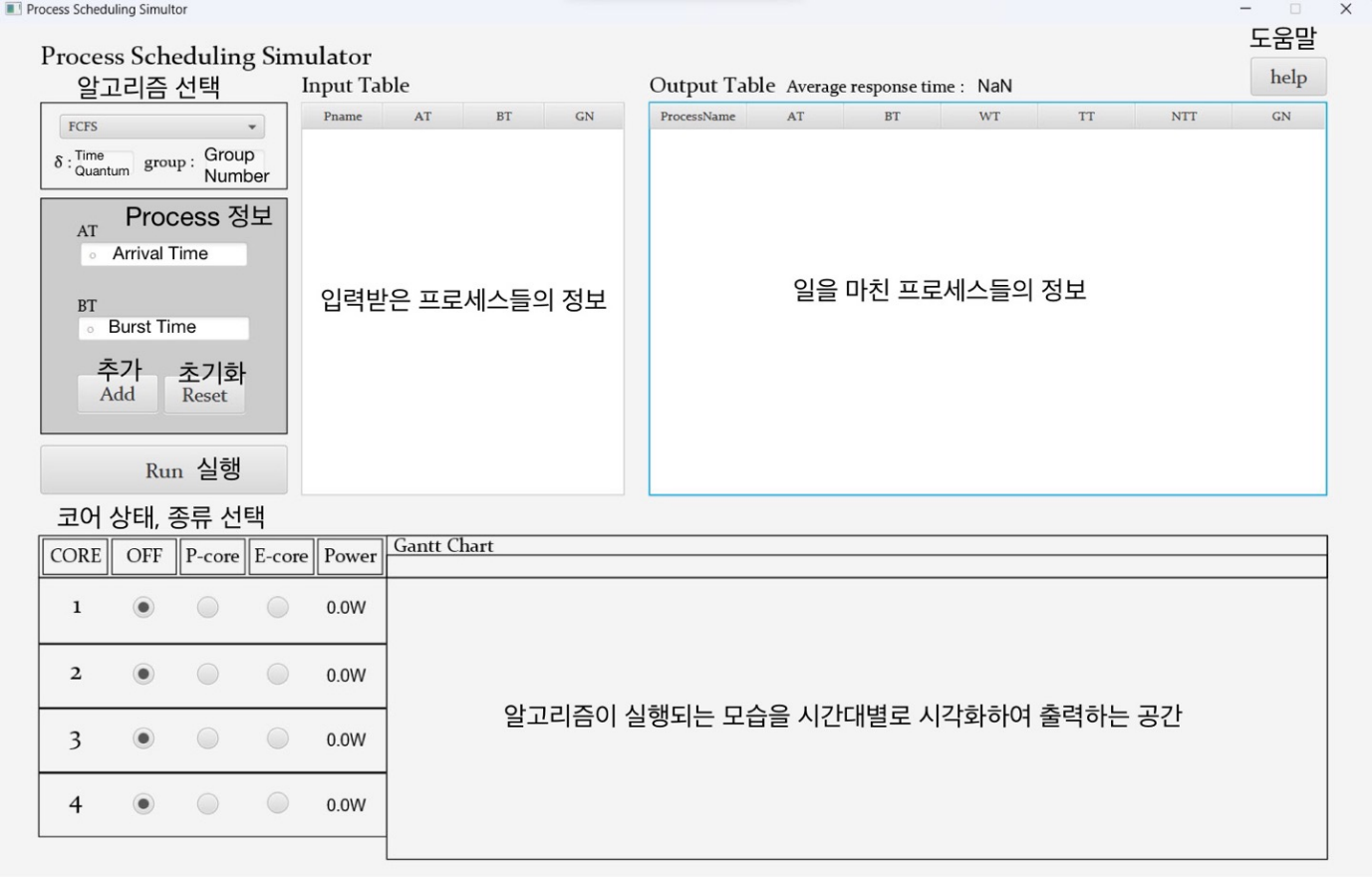
**public void run(Core[] coreList, LinkedList<Process> waitingList, LinkedList<Process> endList, ObservableList<Process> obser, ObservableList<Process> observableList, ScrollPane GanttChart, HBox hBox, Text[] corePower, Text Avgresponsetime, TableView<Process> tableView2, Button[] Buttons,ChoiceBox<String> choicebox)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **필**  **드** | **Data Type** | **Variable Name** | **Value** | **용도 및기능** |
| Core[] | coreList | - | Pcore, Ecore, off 를 받아오는 용도 |
| LinkedList<Process> | waitingList | - | 일을 하기 위해 기다리는 프로세스를 저장하는 용도 |
| LinkedList<Process> | endList | - | 일이 끝난 프로세스를 저장하는 용도 |
| ObservableList<Process> | obser | - | 입력받은 프로세스 리스트 |
| ObservableList<Process> | observableList | - | 패널에 출력을 위한 리스트 |
| ScrollPane | GanttChart | - | 스크롤이있는 패널. 간트차트를 출력하는 용도 |
| Text[] | corePower | - | 코어가 소비한 전력 출력하는 용도 |
| Text | Avgresponsetime | - | 평균 응답시간을 출력하는 용도 |
| TableView<Process> | tableView2 | - | 결과 출력하는 용도 |
| Button[] | Buttons | - | 실행 중 버튼을 잠그는 용도 |
| ChoiceBox<String> | choicebox | - | 실행 중 알고리즘 선택박스를 잠그는 용도 |
| **메소드** | **상세 기능** | ① 입력받은 프로세스의 AT가 현재시간과 같으면 waitingList에 삽입  ② 남은 코어 자리에 waitingList에 있는 프로세스들을 번호 순서대로 할당(GroupNumber가 겹치는 프로세스가 없어서 프로세스를 코어에 할당할 수 있는 경우 할당함)  ③ 간트차트 출력  ④ 코어별로 1초의 일을하고, BT만큼의 일을 Work메소드를 통해 다하면 endList에 삽입  ⑤ endList와 observableList가 같으면 종료  ⑥ 결과 테이블, avgresponseTime 출력 | | |

**3) 코드 동작 과정**

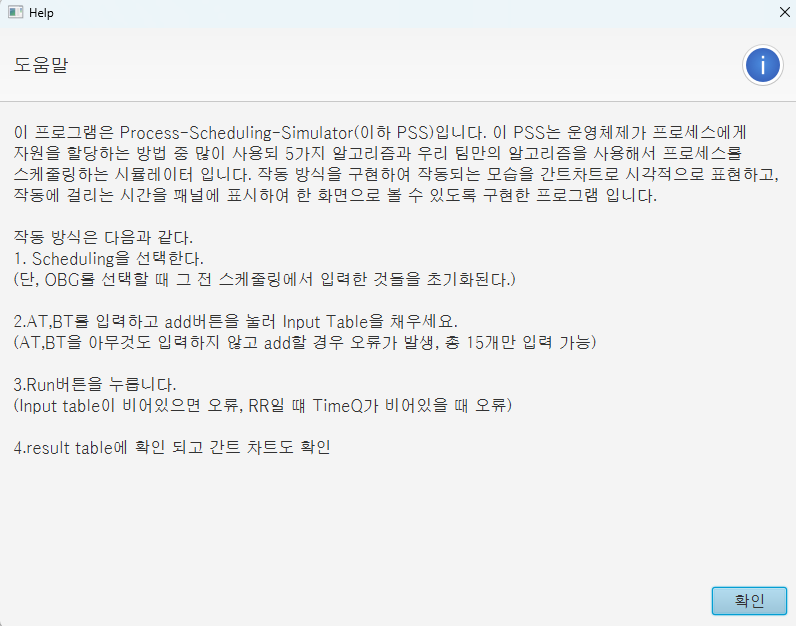
****

**4) 코드 시연**

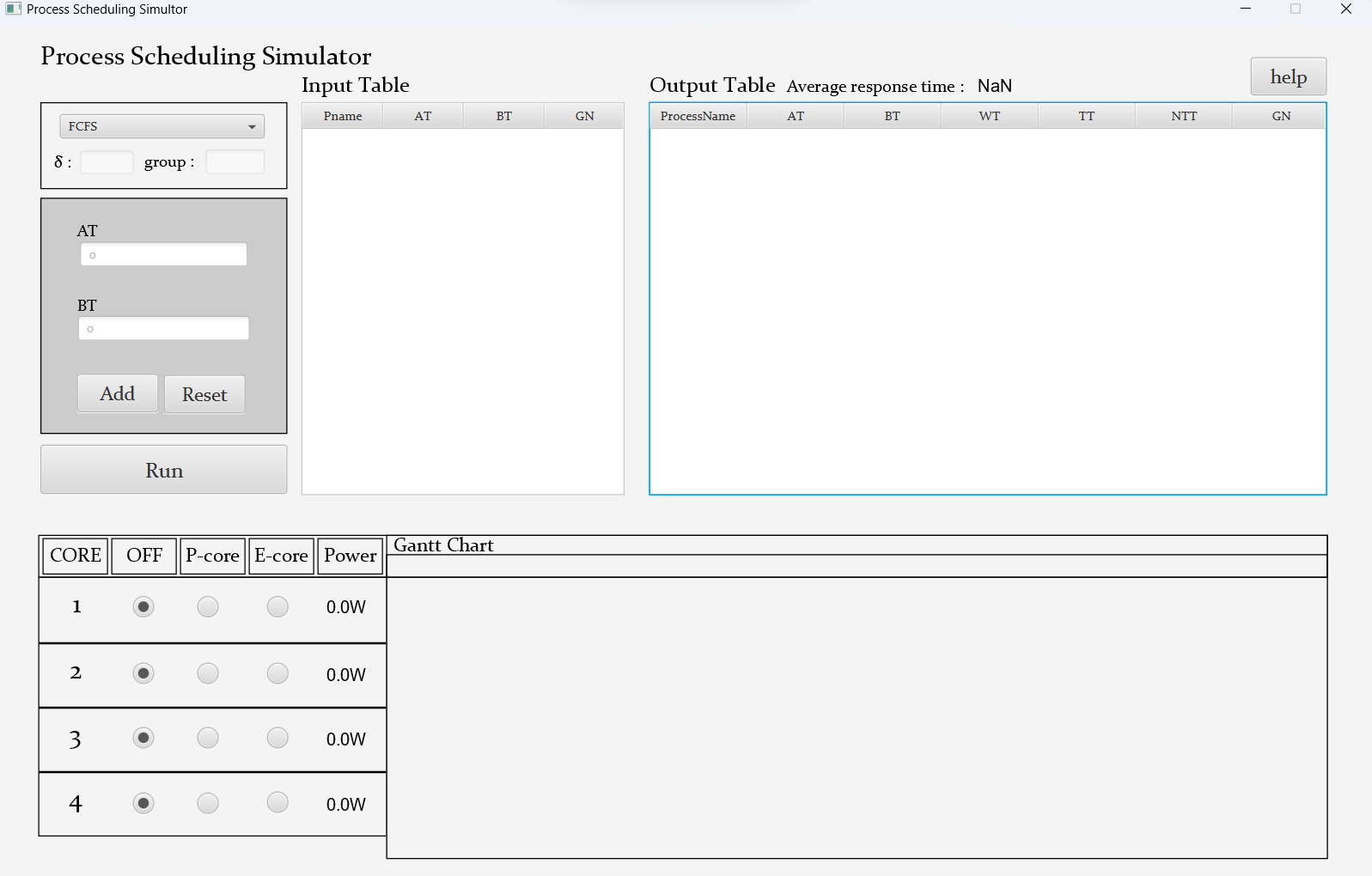


1. 데이터를 입력받는 공간, 출력, 시각화 공간의 구분과 버튼과 체크박스 별 기능

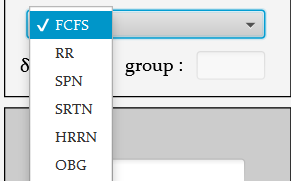
* TimeQuantum은 RR알고리즘에서만, GroupNumber은 OBG알고리즘에서만 입력 가능



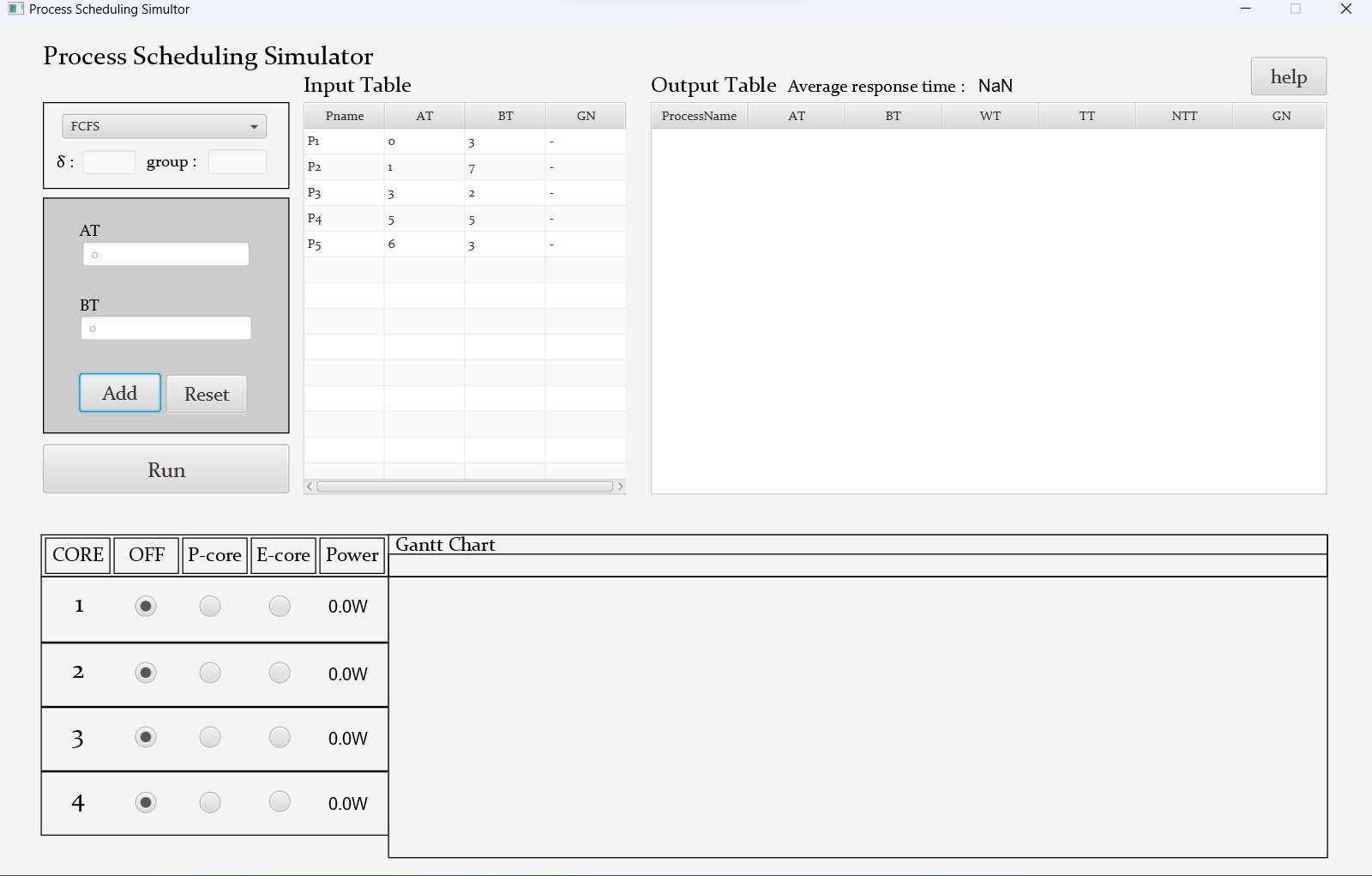
1. 1)도움말 내용

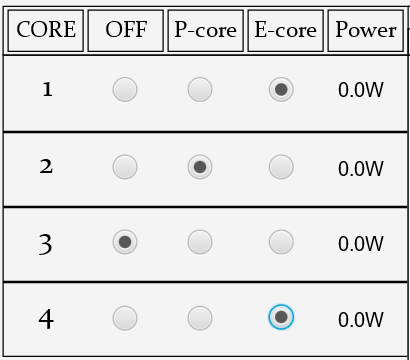
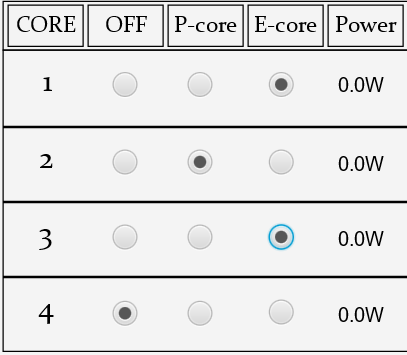
****

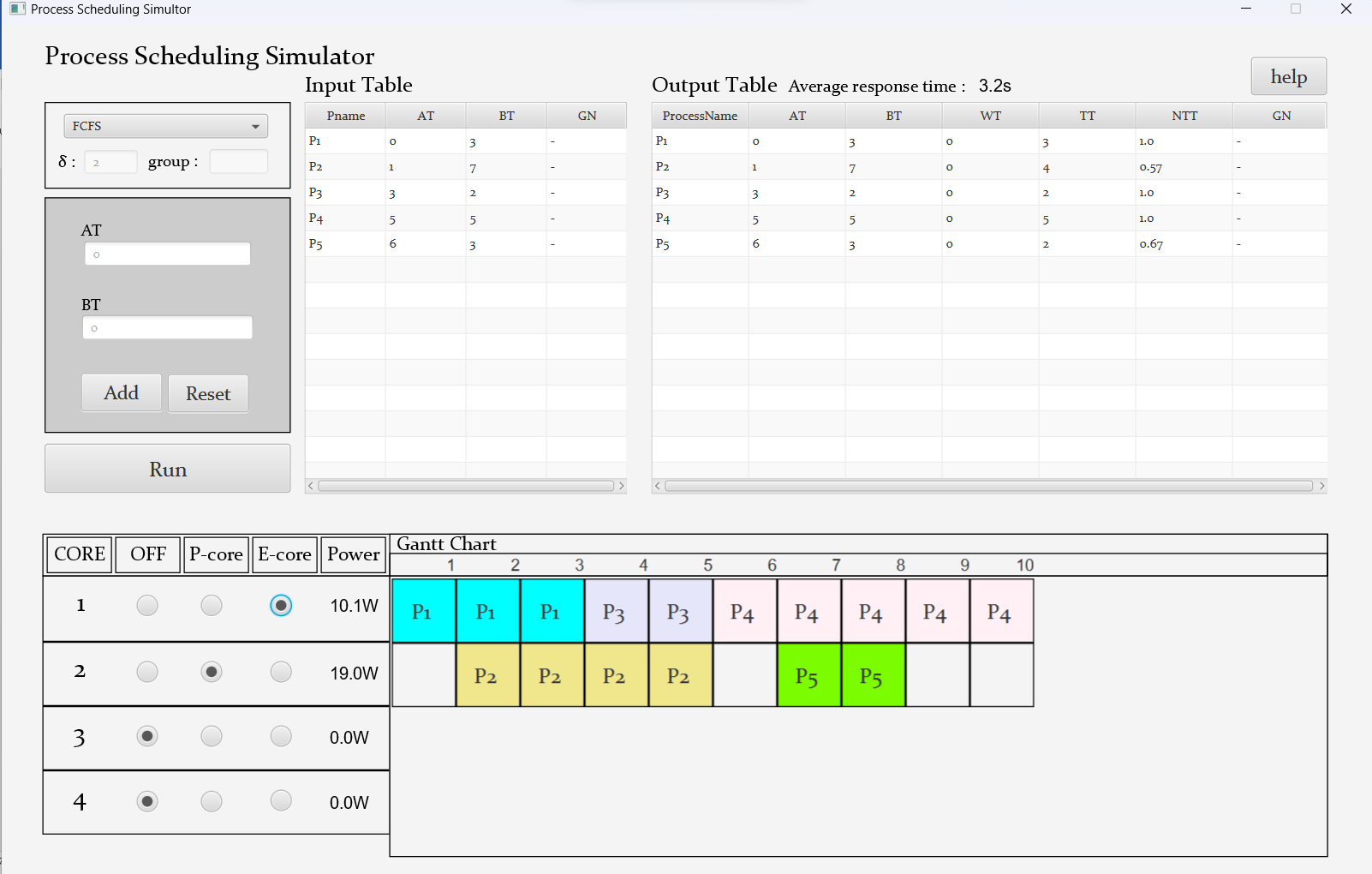
1. 아무 입력을 넣지 않은 상태의 UI 모습



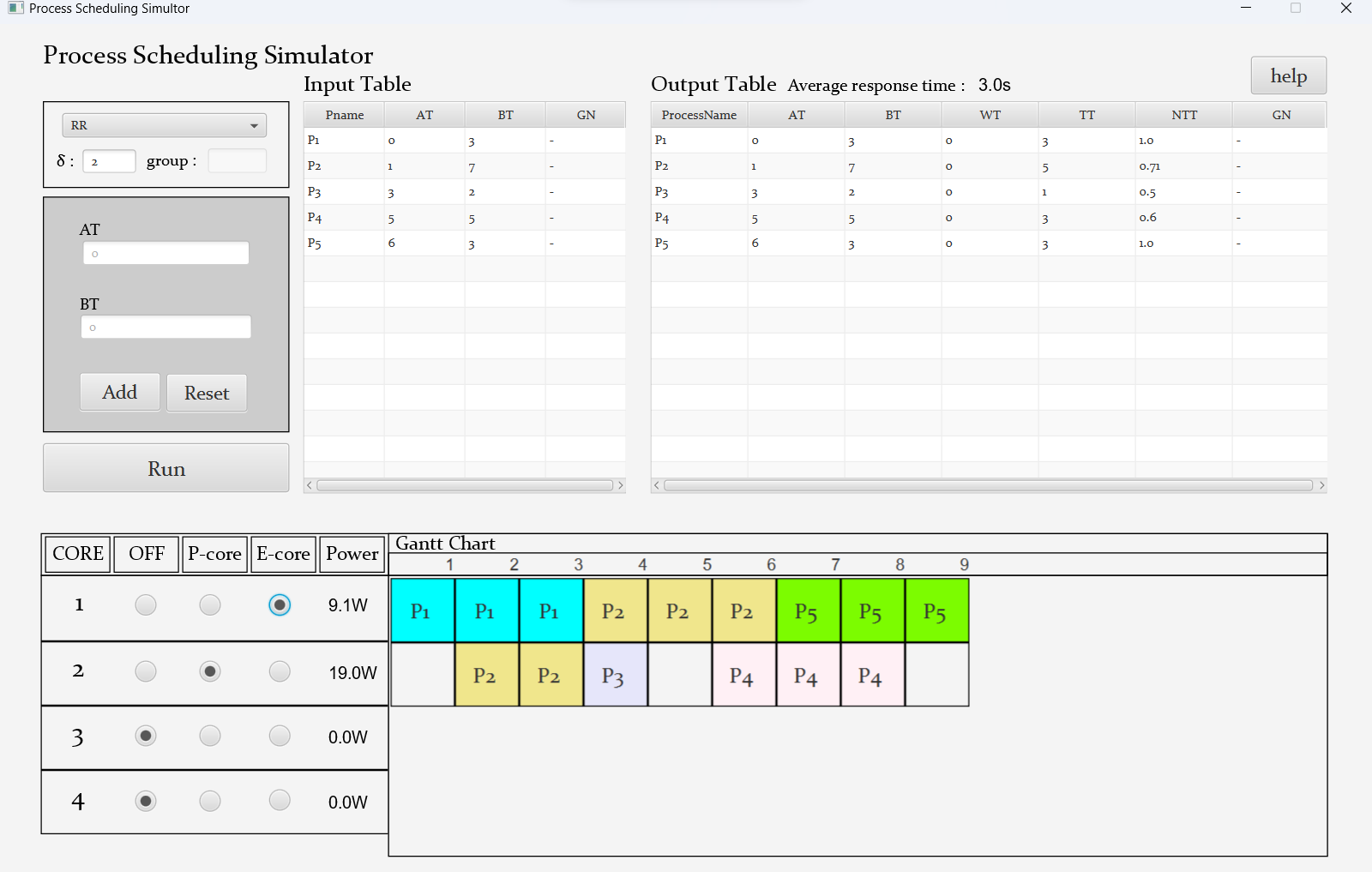
선택박스 클릭시 알고리즘 선택 가능



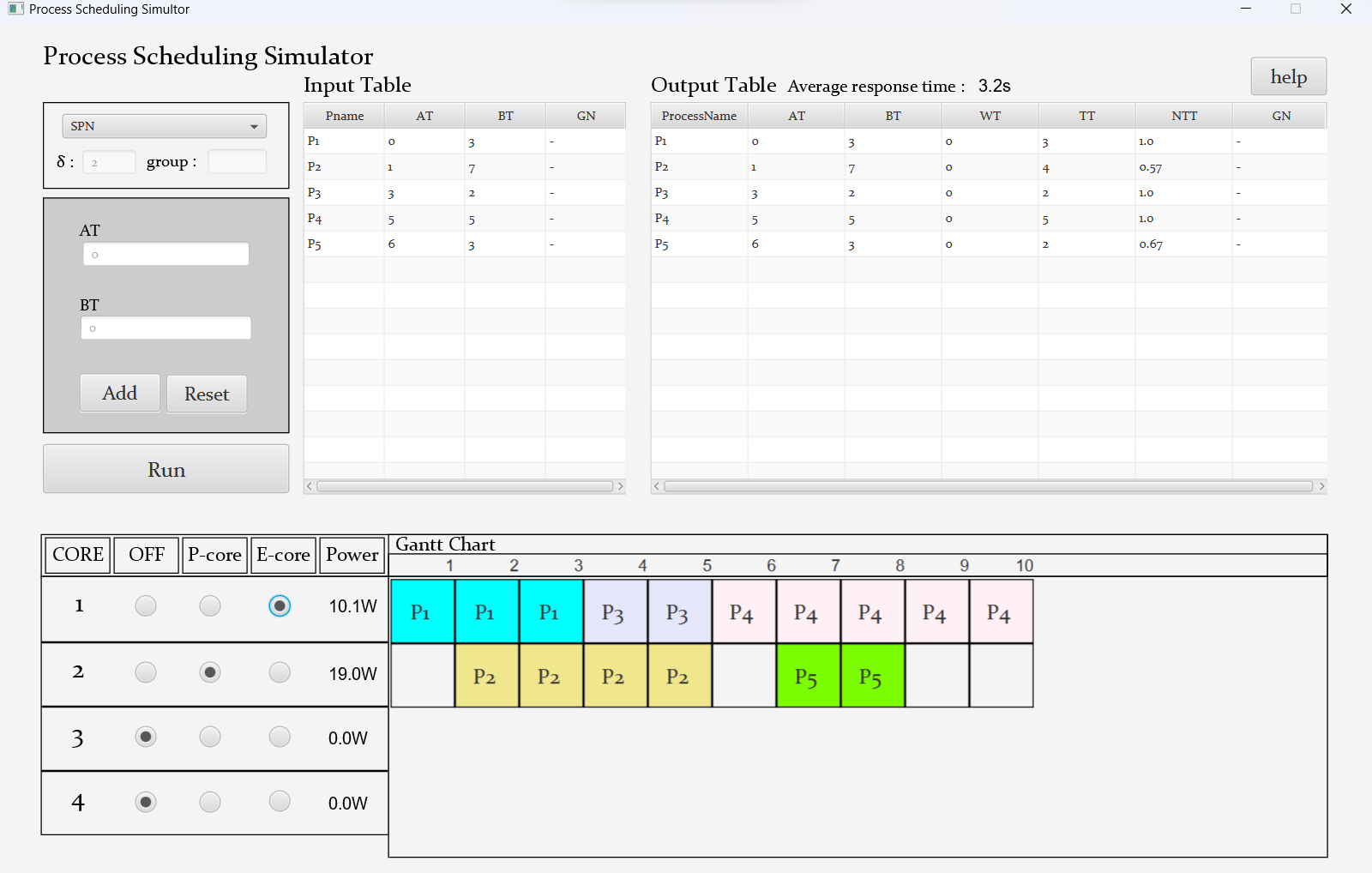
1. 알고리즘을 선택 한 후 데이터 입력 받은 UI 모습
2. 코어(프로세서)별로 OFF, P-core, E-core 선택 가능



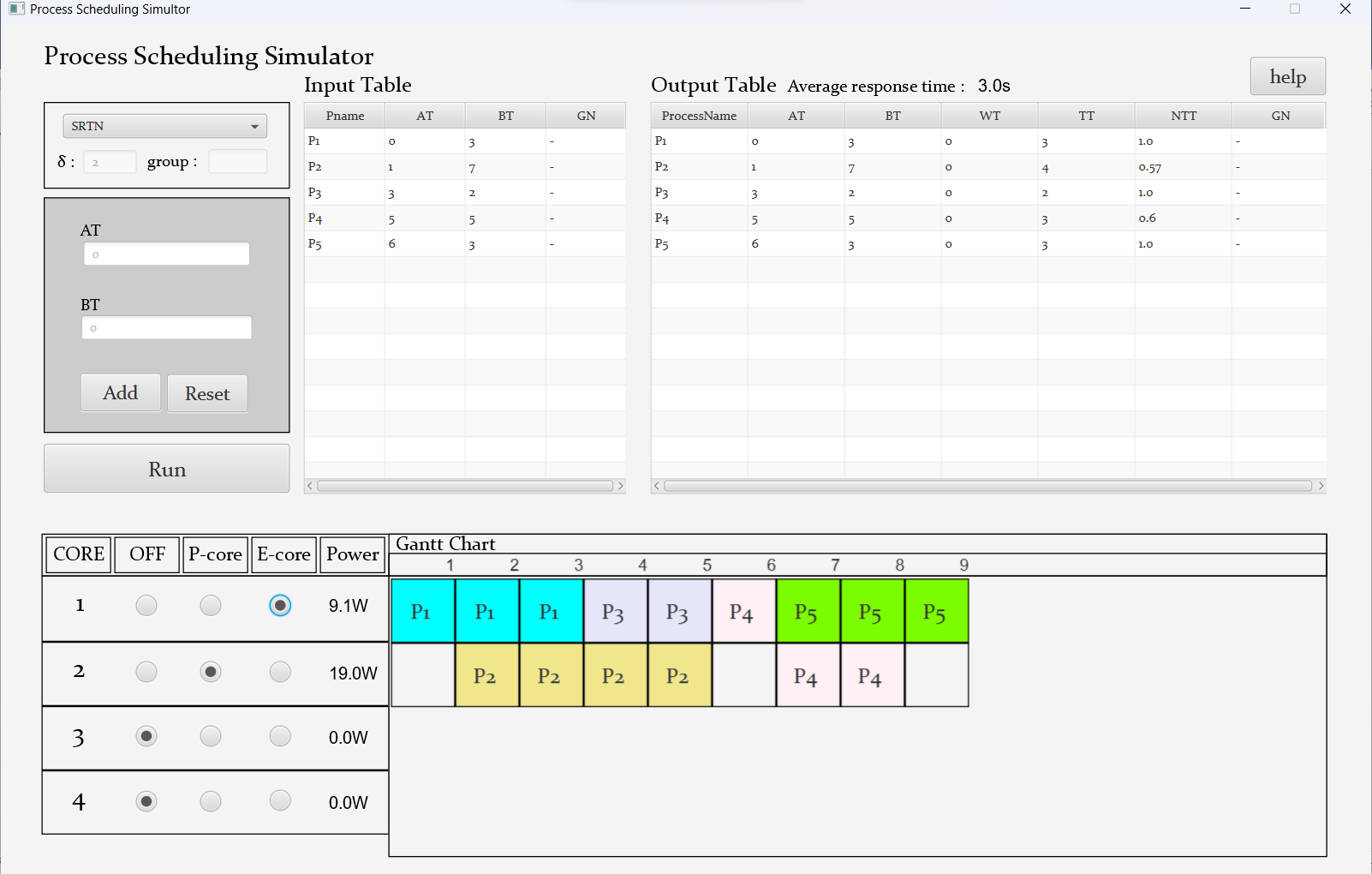
1. 선택한 알고리즘(FCFS) 시뮬레이터 실행 결과



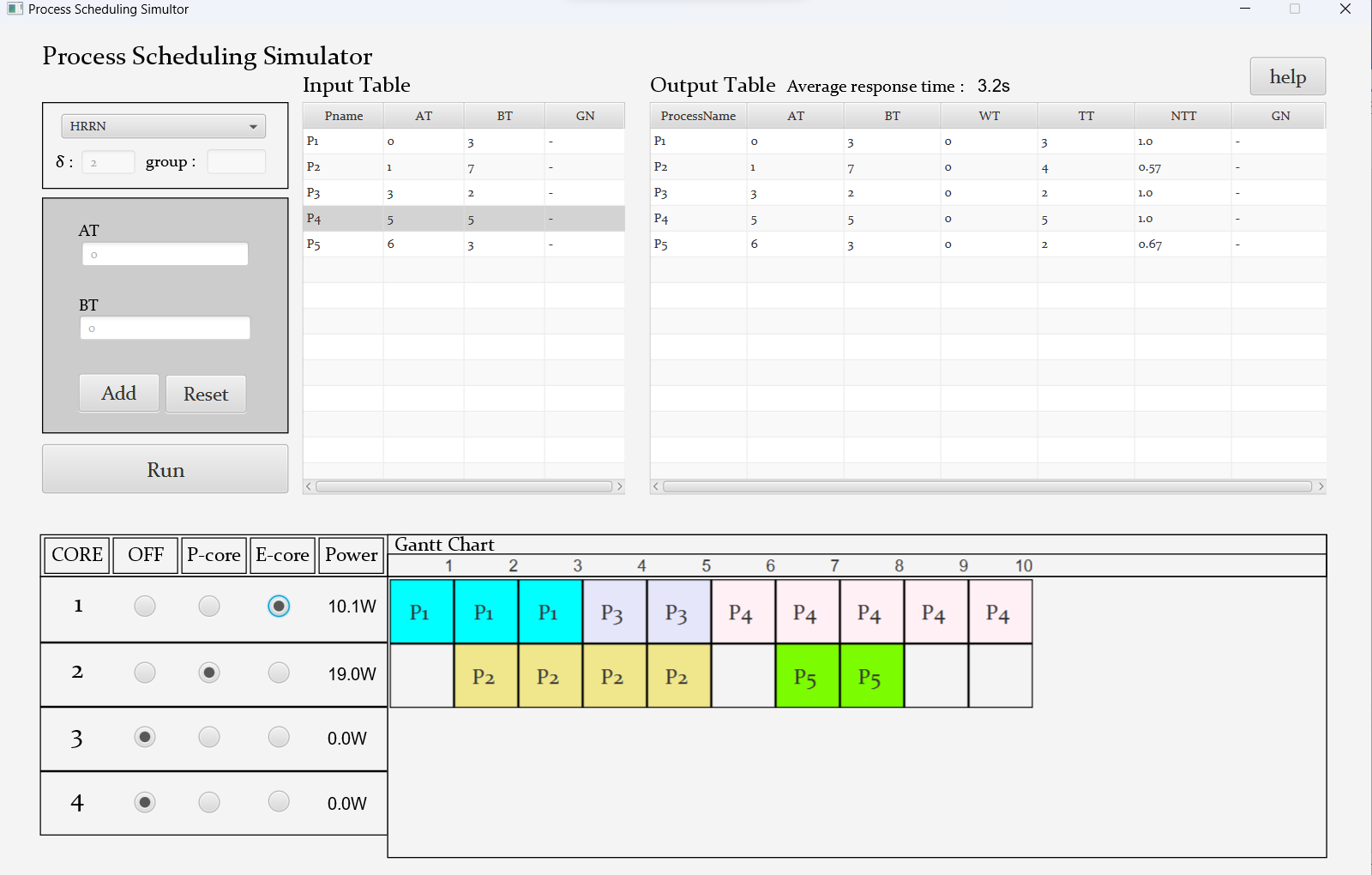
1. 선택한 알고리즘(RR) 시뮬레이터 실행 결과



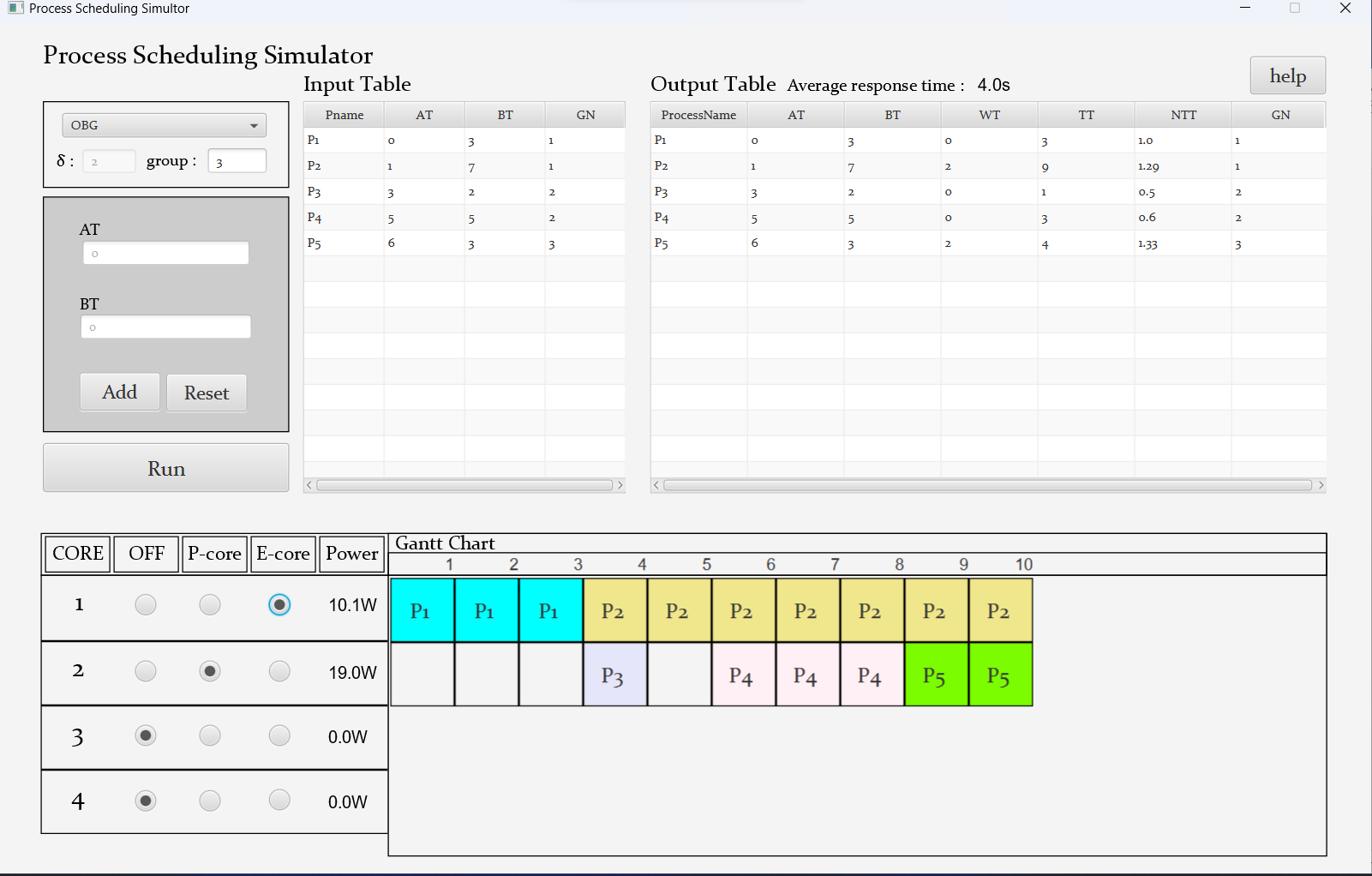
1. 선택한 알고리즘(SPN) 시뮬레이터 실행 결과



1. 선택한 알고리즘(SRTN) 시뮬레이터 실행 결과



1. 선택한 알고리즘(HRRN) 시뮬레이터 실행 결과



1. 선택한 알고리즘(OBG) 시뮬레이터 실행 결과

**결론**

**개인별 성찰 & 소감**

|  |  |
| --- | --- |
| **이름** | **성찰 & 소감** |
| 김형구 | 이번 프로젝트로 인한 여러가지 기술을 배울 수 있어서 좋았습니다. 특히 이번 프로젝트에서 사용한 언어인 java에 있는 javafx를 사용함으로써 어플리케이션을 어떻게 만드는지 알게 되었고 버튼과 마우스의 상호작용이 어떻게 일어나는지 알게 되었으며 다 만들고 나니 많은 버그들이 있었습니다. 예를 들어 at와 bt에 음수가 들어가면 안되는데 들어가게 끔 프로그램이 짜줘있는 버그 같은 것이 있었습니다. 그런 버그를 하나하나 찾고 수정하면서 버그가 얼마나 중요하며 그 버그를 어떻게 수정 할 것인지 또한 버그가 발생할 때 유저에게 어떤 방식으로 전달해야하는지 깊은 고민을 하였습니다. 그리고 이번에 가장 아쉬웠던 점은 버전 관리 프로그램을 사용하지 않는 점이였습니다. 버전 관리 프로그램을 사용하지 않음으로써 여러가지 불편한 상황들이 나왔습니다. 다음에는 버전 관리 프로그램을 사용하는 방향으로 하고 싶습니다. |
| 박금도 | 이전까지 과제를 풀고 시험을 볼 때는 우리가 배웠던 부분 내에서 정답이 있는 코딩만을 해왔었는데, 이렇게 큰 규모의 프로젝트를 해보니 지금까지 배워 온 것들을 활용해야 할 뿐만 아니라 시뮬레이터 개발을 위해 더 신경 써야 할 부분이 많았고 한 문제, 한 문제 개별의 문제가 아닌 하나의 시뮬레이터를 만드는 과정이다 보니 기능 하나를 추가하는 것도 기능 하나를 그냥 덧붙이는 것이 아니라 더 신경 써야 할 부분이 많다는 것을 알게 되었습니다. 그렇다 보니 프로젝트를 진행하며 깃허브에 대한 이야기를 자주 했었습니다. 프로젝트를 시작할 때는 큰 어려움을 느껴본 적이 없어 깃허브의 필요성을 그렇게 크게 느끼지 못했는데, 진행 도중 진행 상황이 제대로 전달되지 않거나 불필요한 중복이 생기기도 하고 예상하지 못한 오류나 생기기도 하는 등 이전에는 경험해 보지 못한 여러 문제와 마주했습니다. 그러는 와중에 RR과 HRRN과 같은 preemptive 알고리즘을 코딩하는 것은 생각만큼 쉽게 해결되지 않고 시간만 허비되어 중간부터는 이 프로젝트가 제대로 완성될 수 있을까 하는 걱정을 많이 했습니다. 고맙게도 다른 조원들이 이런 부분을 보충해 주고 프로젝트에 더 신경 써줬기에 지금은 처음 프로젝트를 받았을 때 생각했었던 결과물보다 더 좋은 결과물이 나올 수 있었습니다. 이번 프로젝트를 통해 제가 부족한 점을 확인할 수 있었고 앞으로 어떤 부분을 더 신경 쓰고 공부해야 하는지 알게 되었습니다. |
| 박상준 | 이때까지는 알고리즘 문제를 풀거나 아무리 커봤자 부모클래스와 자식클래스 정도만 다뤘었는데 이렇게 큰 규모의 프로젝트를 해본 적은 처음이었다. 그래서 그때그때 필요한 코드를 작성해도 코드가 프로젝트에서 정상적으로 동작하도록 적용하는데 많은 시간이 필요했다. 작성하다보니 코드 중복이 많이 발생했는데 줄이는게 굉장히 힘들었고, 또 아직도 많이 줄이지 못한 점이 아쉽다. 코드 관리를 하는데에 있어서도 구조를 잘짜놓으면 유지보수하는데 얼마나 편리한지 느끼게 되었다. 깃허브로 프로젝트를 진행하려고 했는데 사용법을 숙지하는데 시간이 걸릴것 같아서 각자 작업을 한 코드를 export해서 공유했는데 생각보다 너무 불편했다. 깃허브와 같은 버전 관리 툴의 필요성을 절실히 느끼게 되었다. 앞으로 이와 같은 프로젝트를 진행하게 된다면 깃허브를 사용할 생각이다. 실제로 알고리즘 자체를 구현하는데는 오래 걸리지 않았는데 사용자에게 입력과 출력에 대한 UI를 지원하고 사용자 친화적으로 표현하는게 어려웠다. 결국 사용자에게 보여지는 것은 디자인적인 부분이지만 디자인에 대해서는 지식이 없어서 별다른 작업을 하지 못했는데 그 부분이 아쉽다. 규모있는 기업에 디자인부서가 왜 있는지 알 것 같았다. 그래도 직접 짠 코드가 동작하는 것을 보니 뭔가 뿌듯하다. 알고리즘 문제를 풀거나 작은 메소드를 구현하는 것보다 프로젝트 전반을 만들어보니 내가 무엇을 모르고 있었는지 알게 되었고 프로그래밍 능력이 크게 향상된 것 같아 좋았다. |
| 이해민 | 교수님께서 수업시간에 프로젝트에서 대해서 설명을 해주시고 이전 선배님들의 완성본을 보여주셨었는데 처음 봤을 때 프로젝트 제출은 할 수 있을지 걱정이 될 만큼 어렵게 다가왔다. 팀원들과 프로젝트를 시작하고 초반 부분에서는 스케줄링 알고리즘에 대해 어떻게 구현해야 할지에 대한 문제가 많았는데 박상준과 김형구의 첫 시작이 컸다고 생각한다. 어떻게 작성할지 막막하던 상황에서 기초 틀을 잡아주었고 그것에 맞춰 이후 다른 알고리즘들을 구현해 나가는 것은 처음보다는 진전이 있다고 느꼇다. 하지만 여러명이서 나눠서 진행하는 만큼 공유해야될 상황이 매우 많았는데 깃허브를 사용하는게 익숙하지 않아 코드가 바뀔 때 마다 하던 작업을 보내주고 팀원이 보내주면 하던 작업에 합치거나 수정해서 다시 보고 하는 부분이 매우 불편했다. 깃허브를 중요하게 생각하지 않고 있었는데 이번 팀프로젝트를 진행하면서 깃허브에 중요성을 크게 느낄 수 있었다. UI부분은 관여한 부분이 없다시피 하는데 알고리즘을 구현하는 것보다 사용자에게 보여질 UI가 더 중요하다는 생각이 많이 들었다. 알고리즘을 아무리 잘 구현하더라도 UI가 좋지 않다면 사용하기 힘들 것이고 사용자는 줄 것 이다. 프로그램을 만드는 부분에서 UI를 만드는 디자인 부분 일이 중요하다고 생각하게 되었다. 마지막으로 이번 프로젝트를 진행하면서 지금 내가 알고리즘이나 코드를 작성하는 부분에 있어서 어느 정도 수준인지 조금은 알 수 있게 된 것 같고 UI, 깃허브의 중요성 등 여러 부분에서 아직 알아가야 할 부분이 많다는 것을 느끼는 시간들이었다. |