

Operating System LAB 1

CPU Scheduler Simulator

2024. 03. 27

Teaching Assistant : Minguk Choi

Email : mgchoi@dankook.ac.kr

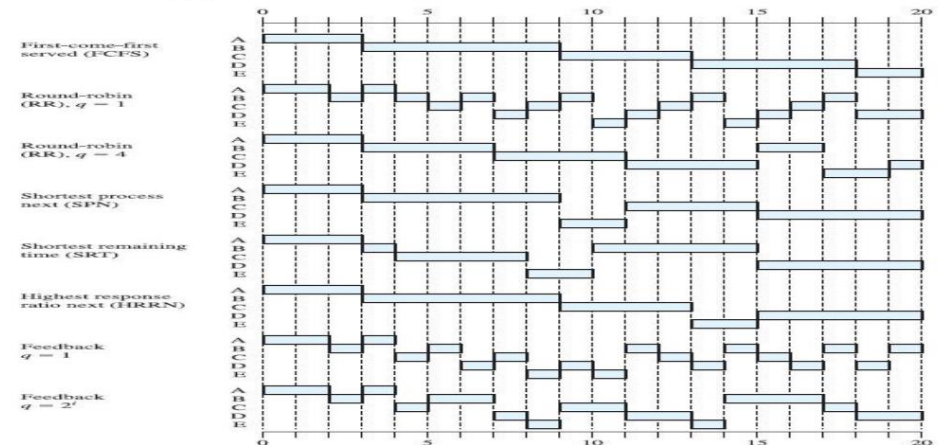
1. 과제 목표

1. CPU Scheduler Simulator 구현한다.
2. Context Switch Time에 따른 Scheduling 결과를 분석한다.
3. 다양한 Workload를 분석하고, 이에 따른 Scheduling 결과를 분석한다.
4. Google Test Framework을 경험한다.

- Workload: 5 processes (jobs)

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- Scheduling policies



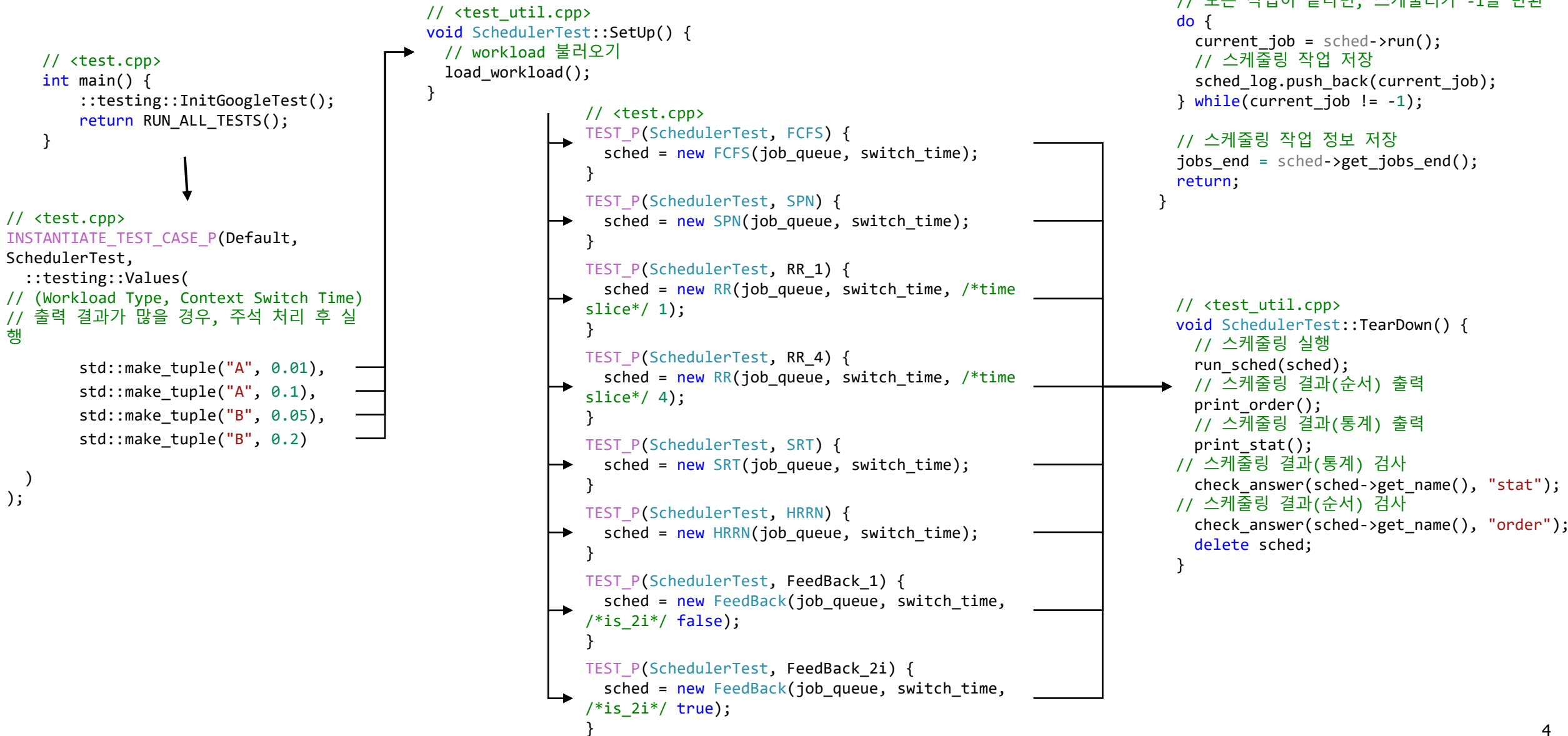
2. 구현 – 환경 구성

- 리눅스 설치 (Lab 0)
 - Virtual Box, WSL
- Lab1 환경 구성
 - OpenSSL, Gtest 설치
 - `sudo apt install libssl-dev`
 - `sudo apt-get install libgtest-dev`
 - Lab1 clone & build & run
 - `git clone https://github.com/DKU-EmbeddedSystem-Lab/2024_DKU_OS`
 - `cd lab1`
 - `make`
 - `./test`

Lab1 폴더 구성

```
├── data
│   ├── answer // 정답 파일 (해시값)
│   │   └── ....
│   ├── workload_A.csv
│   └── workload_B.csv
├── Makefile
├── schd.cpp // scheduler cpp file (구현 후 제출 파일)
├── schd.h // scheduler header file
├── test.cpp // test main file
├── test_util.cpp // test utility cpp file
└── test_util.h // test utility header file
```

2. 구현 – Code Flow



2. 구현 – 부모 Scheduler Class

```
// <schd.h>
// Scheduler 클래스는 모든 스케줄러의 부모 클래스임
class Scheduler {
protected:
    // 스케줄러 이름
    std::string name;
    // workload 작업들이 이름 순으로 정렬된 큐
    std::queue<Job> job_queue_;
    // 작업이 종료된 job을 저장하는 vector
    std::vector<Job> end_jobs_;
    // context_switch 시간 (= 기존 작업 저장 + 새로운 작업 불러오는 시간)
    // switch time은 스케줄링 순서에도 영향을 미치니, 주의해야 함
    double switch_time_;
    // 현재 시간 = 기존 총 작업 실행 시간 + 기존 총 문맥 교환 시간
    // arrival_time, response_time 또한 이를 기준으로 함.
    double current_time_ = 0;
    // 현재 작업 (처음에는 존재하지 않는 job(name=0)으로 초기화되어 있음)
    Job current_job_;

public:
    Scheduler() = default;
    /*
    (1) 생성자 함수
    - 스케줄링 전, 초기화 및 전처리를 담당하는 함수
    - "부모" 생성자는 수정할 수 없음
    - "자식" 생성자는 부모 클래스의 생성자 함수를 "반드시 호출" 해야함
    - "자식" 생성자의 부모 생성자 함수 호출 및 인자는 수정 불가.
    - 그 이외에는 자유롭게 "오버라이딩"하여 작성 가능
    */
    Scheduler(std::queue<Job> jobs, double switch_overhead)
        : job_queue_(jobs), switch_time_(switch_overhead) {
        name = "Default";
    }
```

```
/*
(2) 스케줄링 함수
- 다음 1초 동안 실행할 작업명을 반환함
- 모든 작업이 완료된 경우, "-1"을 반환함
- "부모" run() 함수는 무조건 "-1"을 반환함
- "자식" run() 함수는 각 스케줄러의 정책에 맞게 재작성 (오버라이딩)
- 각 job의 구조체 멤버 변수는 모두 정확하게 기록 되어야함
- 각 job을 완료한 뒤, 완료한 순서대로 "end_jobs_"에 저장(push_back)해야 함
*/
virtual int run() {
    return -1;
}

// (3) 완료된 작업 정보를 반환하는 함수
virtual std::vector<Job> get_jobs_end () final{
    return jobs_end_;
}

// (4) 스케줄러 이름 반환
virtual std::string get_name () final{
    return name;
}
};
```

```
// <schd.h>
struct Job{
    int name = 0; // 작업 이름
    int arrival_time = 0; // 작업 도착 시간
    int service_time = 0; // 작업 소요(= burst) 시간
    int remain_time = 0; // 남은 작업 시간
    (load_workload에서 service_time과 동일하게 초기화)
    double first_run_time = 0.0; // 작업 첫 실행 시간
    double completion_time = 0.0; // 작업 완료 시간
};
```

2. 구현 – (예시) FCFS Class

```
class FCFS : public Scheduler{
public:
    // 자식 클래스 생성자
    FCFS(std::queue<Job> jobs, double switch_overhead) : Scheduler(jobs,
switch_overhead) {
        name = "FCFS";
    }

    // 스케줄링 함수
    int run() override {
        // 할당된 작업이 없고, job_queue가 비어있지 않으면 작업 할당
        if (current_job_.name == 0 && !job_queue_.empty()){
            current_job_ = job_queue_.front();
            job_queue_.pop();
        }

        // 현재 작업이 모두 완료되면
        if(current_job_.remain_time == 0){
            // 작업 완료 시간 기록
            current_job_.completion_time = current_time_;
            // 작업 완료 벡터에 저장
            end_jobs.push_back(current_job_);

            // 남은 작업이 없으면 종료
            if (job_queue_.empty()) return -1;

            // 새로운 작업 할당
            current_job_ = job_queue_.front();
            job_queue_.pop();
            // context switch 타임 추가
            current_time_ += switch_time_;
        }
    }
};
```

```
        // 현재 작업이 처음 스케줄링 되는 것이라면
        if (current_job_.service_time == current_job_.remain_time){
            // 첫 실행 시간 기록
            current_job_.first_run_time = current_time_;
        }

        // 현재 시간 ++
        current_time_++;
        // 작업의 남은 시간 --
        current_job_.remain_time--;

        // 스케줄링할 작업명 반환
        return current_job_.name;
    }
};
```

2. 구현 – 문제 Scheduler Class

```
class SPN : public Scheduler{
private:
    /*
     * 구현 (멤버 변수/함수 추가 및 삭제 가능)
     */
public:
    SPN(std::queue<Job> jobs, double switch_overhead) : Scheduler(jobs,
switch_overhead) {
        name = "SPN";
        /*
         * 위 생성자 선언 및 이름 초기화 코드 수정하지 말것.
         * 나머지는 자유롭게 수정 및 작성 가능
         */
    }

    int run() override {
        /*
         * 구현
         */
        return current_job_.name;
    }
};
```

```
class RR : public Scheduler{
private:
    int time_slice_;
    int left_slice_;
    std::queue<Job> waiting_queue;
    /*
     * 구현 (멤버 변수/함수 추가 및 삭제 가능)
     */
public:
    RR(std::queue<Job> jobs, double switch_overhead, int time_slice) :
Scheduler(jobs, switch_overhead) {
        name = "RR_"+std::to_string(time_slice);
        /*
         * 위 생성자 선언 및 이름 초기화 코드 수정하지 말것.
         * 나머지는 자유롭게 수정 및 작성 가능 (아래 코드 수정 및 삭제 가능)
         */
        time_slice_ = time_slice;
        left_slice_ = time_slice;
    }

    int run() override {
        /*
         * 구현 (아래 코드도 수정 및 삭제 가능)
         */
        return current_job_.name;
    }
};
```

2. 구현 – 문제 Scheduler Class

```
class SRT : public Scheduler{
private:
    /*
     * 구현 (멤버 변수/함수 추가 및 삭제 가능)
     */
public:
    SRT(std::queue<Job> jobs, double switch_overhead) : Scheduler(jobs,
switch_overhead) {
        name = "SRT";
        /*
         * 위 생성자 선언 및 이름 초기화 코드 수정하지 말것.
         * 나머지는 자유롭게 수정 및 작성 가능
         */
    }

    int run() override {
        /*
         * 구현
         */
        return current_job_.name;
    }
};
```

```
class HRRN : public Scheduler{
private:
    /*
     * 구현 (멤버 변수/함수 추가 및 삭제 가능)
     */
public:
    HRRN(std::queue<Job> jobs, double switch_overhead) : Scheduler(jobs,
switch_overhead) {
        name = "HRRN";
        /*
         * 위 생성자 선언 및 이름 초기화 코드 수정하지 말것.
         * 나머지는 자유롭게 수정 및 작성 가능
         */
    }

    int run() override {
        /*
         * 구현
         */
        return current_job_.name;
    }
};
```


2. 구현 – 문제 Scheduler Class

```
// FeedBack 스케줄러 (queue 개수 : 4 / boosting 없음)
class FeedBack : public Scheduler{
private:
    /*
     * 구현 (멤버 변수/함수 추가 및 삭제 가능)
     */
public:
    FeedBack(std::queue<Job> jobs, double switch_overhead, bool is_2i) :
    Scheduler(jobs, switch_overhead) {
        if(is_2i){
            name = "FeedBack_2i";
        } else {
            name = "FeedBack_1";
        }

        /*
         * 위 생성자 선언 및 이름 초기화 코드 수정하지 말것.
         * 나머지는 자유롭게 수정 및 작성 가능
         */
    }

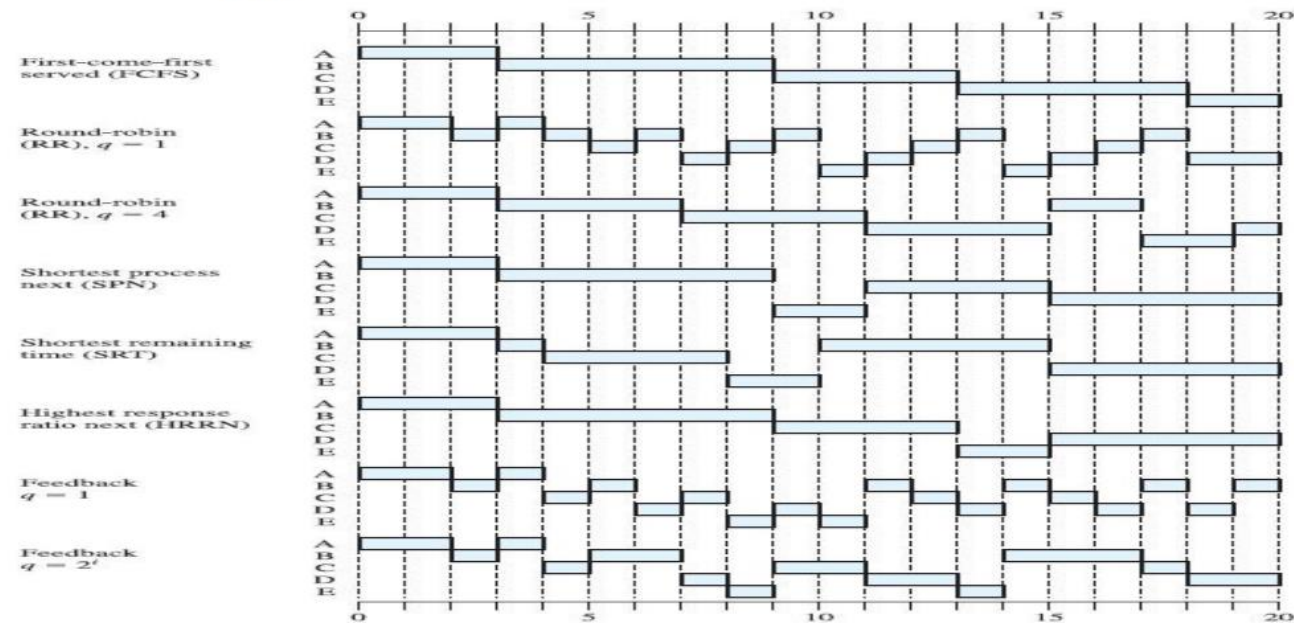
    int run() override {
        /*
         * 구현
         */
        return current_job_.name;
    }
};
```

2. 구현 – Workload A

- Workload: 5 processes (jobs)

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- Scheduling policies



+ Context Switch Time : 0.01 / 0.1

2. 구현 – Workload B

Process	Arrival Time	Service Time
1	0	10
2	1	3
3	2	2
4	6	5
5	11	12
6	15	2
7	18	4
8	20	3

+ Context Switch Time : 0.05 / 0.2

2. 구현 - 문제 / 입력

■ 문제

- 스케줄링 알고리즘에 따라 프로세스를 정확한 순서대로 스케줄링하고, 모든 작업 정보를 정확히 기록하는 스케줄러를 작성하라.
- 스케줄러 종류는 다음과 같다 : RR, SPN, SRT, HRRN, FeedBack.
- 스케줄러 정책은 LN3_Scheduling.pdf의 23page의 그림에 따른다.
- 스케줄러는 생성자를 통해 실행할 작업들이 저장된 job_queue와 context switch time을 전달 받는다.
- 스케줄러는 run()함수가 호출될 때마다, 다음 1초간 실행할 작업명을 반환한다.
- 스케줄러의 run()함수는 모든 작업이 완료된 경우, -1을 반환한다.
- 스케줄러는 job 구조체의 정보가 변경될 때마다, 이를 모두 update한다.
- 스케줄러는 완료된 job을 "end_jobs"에 순서대로 저장 (push_back)한다.

■ 입력 (Workload)

- 프로세스 (Job)의 개수 (n): $0 \leq n \leq 10$ (개)
 - Job_queue에는 작업들이 arrival time순으로 정렬되어 저장되어 있다.
- 각 프로세스 Service Time (s) : $1 \leq s \leq 100$
- 각 프로세스 Arrival Time (a) : $1 \leq a \leq 100$
- 문맥 교환 (Context Switch) Time (c) : $0.01 \leq c \leq 1$
- Test workload 수행 시, CPU는 작업이 완료될 때까지 idle한 경우가 존재하지 않는다.

2. 구현 – 구현/채점

■ 구현

- schd.cpp의 RR, SPN, SRT, HRRN, FeedBack 클래스를 구현한다.
 - 각 클래스의 1) 생성자, 2) int run() 함수를 수정 및 작성하여 구현한다.
 - 각 클래스의 위 2가지 함수의 선언은 수정할 수 없다.
 - 생성자는 부모 생성자를 호출하고, name을 초기화 해야한다. (기존 내용을 수정하지 말 것)
 - 각 클래스의 멤버 변수/함수는 자유롭게 추가 가능하다.
 - 반드시 C++로 구현해야 한다.
 - 라이브러리는 C++ STL만 사용 가능하다.
 - schd.cpp 외 다른 파일은 수정, 추가, 제출이 불가하다.
- RR과 FeedBack은 time quantum이 다르더라도, 동일한 class로 작성한다.
 - 생성자를 통해 time quantum을 전달 받는다.
- FeedBack의 큐 개수는 4개, Boosting 정책은 없다.

■ 채점

- 학생이 제출한 "schd.cpp"는 Makefile를 통해, 다른 소스코드와 "test" 프로그램으로 build 되어야 한다.
- "test"프로그램은 입력 받은 Test Case(workload)를 수행하여, 1) 작업 스케줄링 순서, 2) 완료된 작업들의 정보를 정확하게 출력한다.
- Known Case는 make 후, test를 실행하여 학생들이 직접 채점결과를 확인할 수 있다.

2. 구현 - 채점 화면

```

mingu@server:~/lab1_sch$ ./test
[=====] Running 16 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 16 tests from Default/SchedulerTest
[ RUN      ] Default/SchedulerTest.FCFS/0
-----
Process | 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
-----
P0      | [] [] []
P1      |
P2      |      [] [] [] [] []
P3      |          [] [] [] []
P4      |              [] [] [] [] []

-----
Name | Arrival Time | Service Time | First Run Time | Completion Time | Turn Around Time | Response Time
-----
P1   | 0             | 3             | 0               | 3               | 3               | 0
P2   | 2             | 6             | 3.01            | 9.01            | 7.01            | 1.01
P3   | 4             | 4             | 9.02            | 13.02           | 9.02            | 5.02
P4   | 6             | 5             | 13.03           | 18.03           | 12.03           | 7.03
P5   | 8             | 2             | 18.04           | 20.04           | 12.04           | 10.04
-----
AVG  | 4             | 4             | 8.62            | 12.62           | 8.62            | 8.62
-----
[ OK      ] Default/SchedulerTest.FCFS/0 (0 ms)
[ RUN      ] Default/SchedulerTest.FCFS/1
-----
Process | 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
-----
P0      | [] [] []
P1      |      [] [] [] [] []
P2      |          [] [] [] []
P3      |              [] [] [] []
P4      |                  [] []

-----
Name | Arrival Time | Service Time | First Run Time | Completion Time | Turn Around Time | Response Time
-----
P1   | 0             | 3             | 0               | 3               | 3               | 0
P2   | 2             | 6             | 3.1             | 9.1             | 7.1             | 1.1
P3   | 4             | 4             | 9.2             | 13.2            | 9.2             | 5.2
P4   | 6             | 5             | 13.3            | 18.3            | 12.3            | 7.3
P5   | 8             | 2             | 18.4            | 20.4            | 12.4            | 10.4
-----
AVG  | 4             | 4             | 8.8             | 12.8            | 8.8             | 8.8
-----
[ OK      ] Default/SchedulerTest.FCFS/1 (0 ms)

```

출력 결과가 많을 경우, 아래와 같이 주석 처리 후 실행

```

// <test.cpp:41>
INstantiateTest_CASE_P(Default, SchedulerTest,
::testing::Values(
    // std::make_tuple("워크로드 파일", "context_switch 시간"),
    std::make_tuple("A", 0.01)
    // std::make_tuple("A", 0.1),

    // std::make_tuple("B", 0.01),
    // std::make_tuple("B", 0.2)
)
);

```

```

[-----] Global test environment tear-down
[=====] 16 tests from 1 test suite ran. (1 ms total)
[ PASSED ] 2 tests.
[ FAILED ] 14 tests, listed below:
[ FAILED ] Default/SchedulerTest.RR_1/0, where GetParam() = ("A", 0.01)
[ FAILED ] Default/SchedulerTest.RR_1/1, where GetParam() = ("A", 0.1)
[ FAILED ] Default/SchedulerTest.RR_2/0, where GetParam() = ("A", 0.01)
[ FAILED ] Default/SchedulerTest.RR_2/1, where GetParam() = ("A", 0.1)
[ FAILED ] Default/SchedulerTest.SPN/0, where GetParam() = ("A", 0.01)
[ FAILED ] Default/SchedulerTest.SPN/1, where GetParam() = ("A", 0.1)
[ FAILED ] Default/SchedulerTest.SRT/0, where GetParam() = ("A", 0.01)
[ FAILED ] Default/SchedulerTest.SRT/1, where GetParam() = ("A", 0.1)
[ FAILED ] Default/SchedulerTest.HRRN/0, where GetParam() = ("A", 0.01)
[ FAILED ] Default/SchedulerTest.HRRN/1, where GetParam() = ("A", 0.1)
[ FAILED ] Default/SchedulerTest.FeedBack_1/0, where GetParam() = ("A", 0.01)
[ FAILED ] Default/SchedulerTest.FeedBack_1/1, where GetParam() = ("A", 0.1)
[ FAILED ] Default/SchedulerTest.FeedBack_2i/0, where GetParam() = ("A", 0.01)
[ FAILED ] Default/SchedulerTest.FeedBack_2i/1, where GetParam() = ("A", 0.1)

```

14 FAILED TESTS

3. 보고서

- 보고서 구성

1. 구현 설명

- ① 구현한 소스코드 설명

2. Discussion

- ① 여러 워크로드에서의, 다양한 스케줄러의, 서로 다른 지표에 의한 분석

- **a) Workload A/B에서 b) Switch Time이 0.01/0.05/0.1/0.2일 때, c) Average/Worst d) Turnaround/Response Time이 가장 e) 낮은/높은 정책은 무엇인가? 그 이유를 스케줄링 규칙과 실험 결과를 바탕으로 설명하시오.**

- 자신이 관심있는 3~4 가지 기준으로 작성 (Workload 분석, 그래프 작성 추천)

- ② 과제를 하면서, 새롭게 배운 점 / 어려웠던 점

...

자유롭게 작성

4. 과제 제출

- 과제 제출 링크 (구글 폼)
 - <https://forms.gle/tjKbqLhZdZat84E8A>
- 과제 제출 기한 : 24년 4월 10일 수요일 23:59:59
 - 제출 마감 전, 답변 수정 가능 (중복 제출X)
- 과제 제출 목록
 1. 보고서
 - 파일명 : os_lab1_학번_이름.pdf
 2. 소스코드
 - 제출형식: sched_학번_이름.cpp
 - sched.cpp 상단에 학번과 이름 주석으로 반드시 작성

4. 과제 제출

[Operating System] LAB 1

2024 단국대
Operating System LAB 1

koreachoi96@gmail.com [계정 전환](#)

파일을 업로드하고 이 양식을 제출하면 Google 계정과 연결된 이름, 이메일 주소 및 사진이 기록됩니다.

* 표시는 필수 질문임

이메일 *

☒ koreachoi96@gmail.com을(를) 내 응답에 포함할 이메일로 기록합니다.

개인정보 수집 · 이용 동의서

이와 관련하여 아래와 같이 귀하의 개인정보를 수집 및 이용 내용을 개인정보보호법 제 15조 (개인정보의 수집 · 이용) 및 통계법 33조 (비밀의 보호 등)에 의거하여 안내 드리오니 확인하여 주시기 바랍니다.

- 개인정보 수집 이용의 목적: 2024 단국대 운영체제 수업

- 수집하려는 개인정보의 필수 항목: 성명, 학번, 학과, 학년, 이메일

- 개인정보의 보유 및 이용 기간:

2024 단국대 운영체제 수업을 위한 목적 달성 후 1개월 이내 폐기

개인정보보호법에 의거하여 개인정보 수집 및 이용에 따른 동의를 거부할 수 있으나, 동의를 거부할 경우 수업의 정상적인 참여가 어려울 수 있습니다.

개인정보 수집 및 이용에 동의하십니까?

☒ 동의함

☐ 동의하지 않음

선택해제

다음

1/5페이지

양식 지우기

개인정보

이름 *

최민국

학번 *

32000000

분반 *

☐ 1분반 (최종무 교수님 / 월9,10,11/수9,10,11)

☐ 2분반 (최종무 교수님 / 월12,13,14/수12,13,14)

뒤로

다음

2/5페이지

양식 지우기

1. 보고서 제출

과제 보고서 (파일명 : os_lab1_학번_이름.pdf) *

[파일 추가](#)

뒤로

다음

3/5페이지

양식 지우기

2. 구현 코드 제출

구현 코드 (파일명 : sched_학번_이름.cpp) *

[파일 추가](#)

뒤로

다음

4/5페이지

양식 지우기

17

5. 채점 기준

- 구현은 Google Test를 통해, 모두 자동 채점함
 - Test Case = Known Case (Workload A/B) + Hidden Case
- 총점 100점 = 구현 60점 + 보고서 30점 + 형식 10점
 - 형식 + 보고서 + SPN 구현 = 50점 → 기본 점수 높음
 - 보고서의 구현 설명은 "자신이 구현한 부분까지만 잘 설명"하면 만점
- 감점 사항
 - 지각 제출 시, 하루에 10% 감점
 - 소스코드를 텍스트/이미지/Makefile 빌드 불가 파일로 제출 시 → 구현+형식 점수 (0/70점)
 - 인적사항 주석 미 작성, 파일명/형식 미 준수 → 형식 점수 (0/10점)
 - 코드 표절 시, 관련 학생 모두 일괄 0점 처리 (Moss/Codequiry Program 검사 예정)
 - if/else문 순서 변경, 함수 위치 변경, 변수 명 변경 모두 표절에 해당함
 - GPT 사용가능, 하지만 이로 인한 표절은 모두 제출자 책임

구분	세부사항	점수
구현	SPN	10
	RR(q=1)	8
	RR(q=4)	8
	SRT	8
	HRRN	8
	Feedback(q=1)	8
	Feedback(q=2^i)	10
보고서	구현내용 설명	15
	Discussion	15
공통	형식 준수	10

5. 채점 기준

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

```
if (number == 0) {
    execvp(tokens[0], tokens);
    exit(0);
}

else {
    waitpid(pid, &stat, 0);
}

memset(tokens, '\0', MAX);
return 1;
}

int main() {
    char line[MAX];

    while (1) {
        printf("%s$ ", get_current_dir_name());
        fgets(line, sizeof(line) - 1, stdin);

        if (strcmp("exit", line, 4) == 0) {
            printf("exit mysh.c\n");
            break;
        }
        run(line);
    }
}
```

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

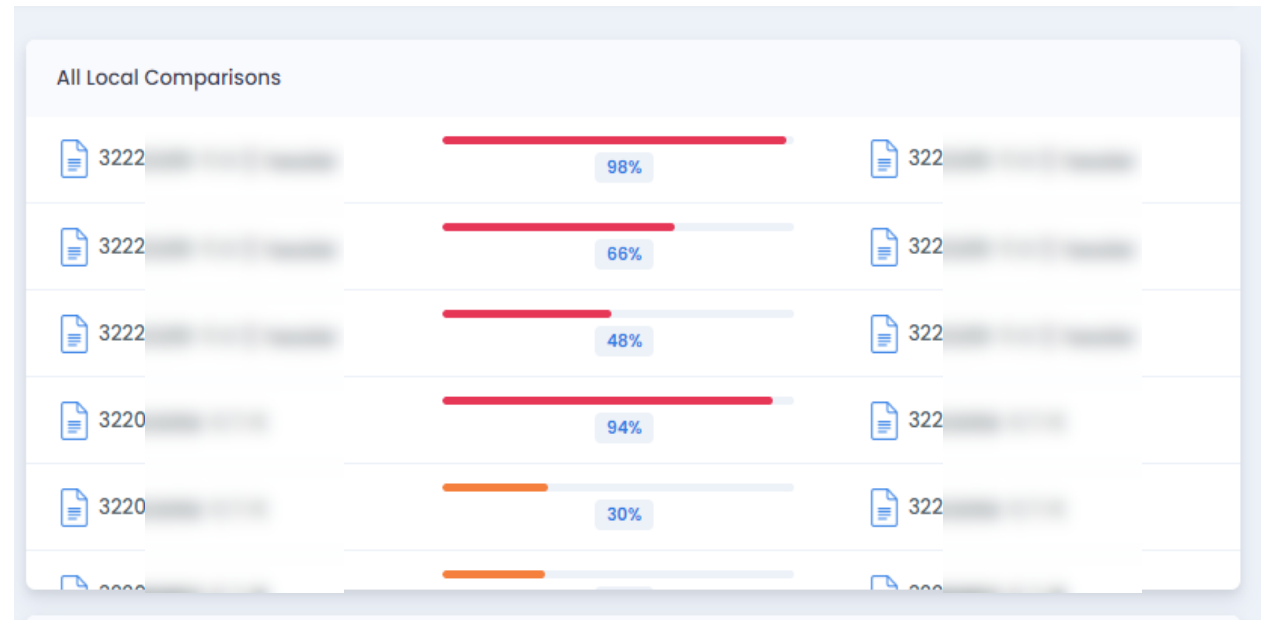
153

154

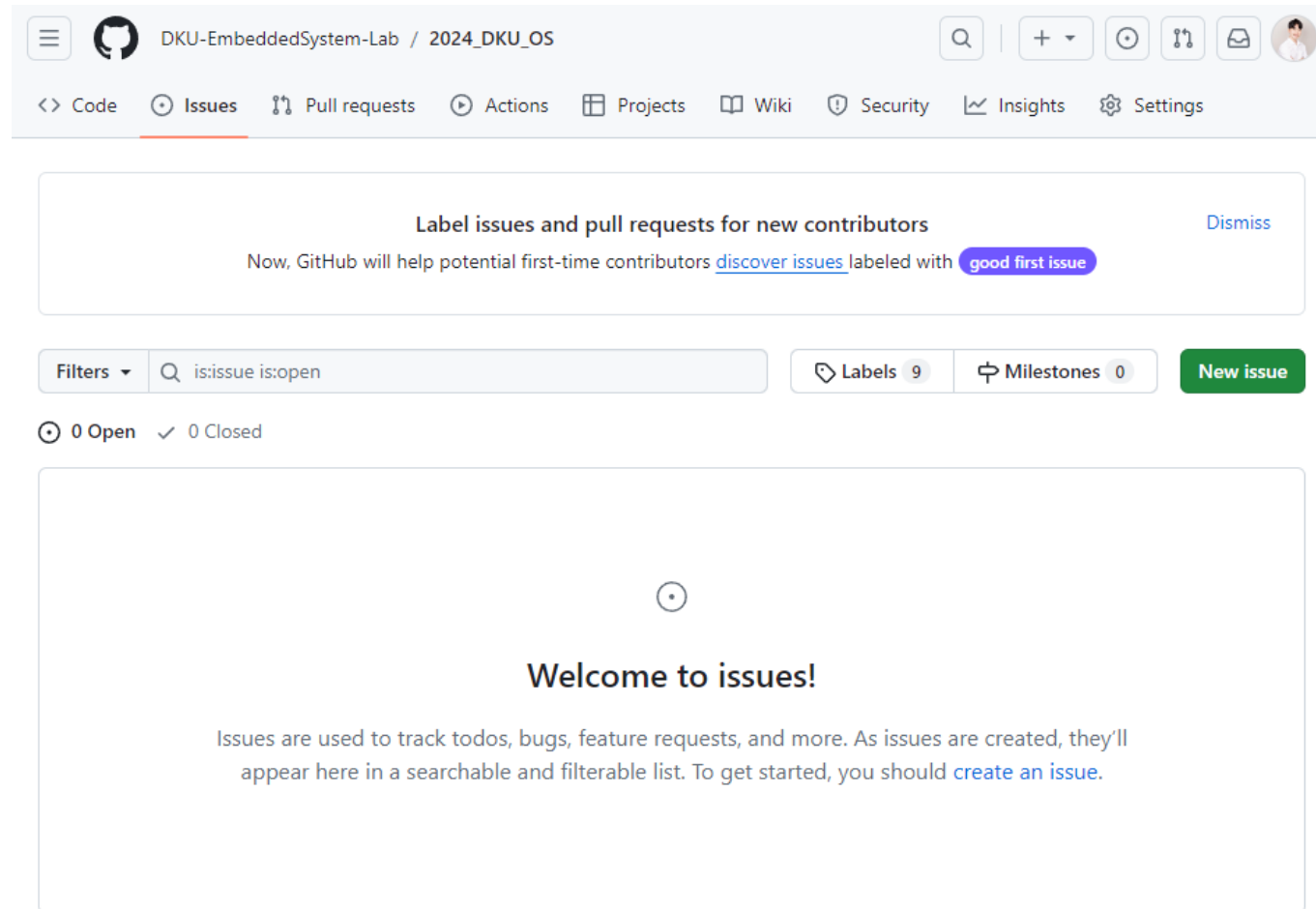
155

156

94%



6. 질문



Please direct **all questions** related to assignments to Github Issues.
(https://github.com/DKU-EmbeddedSystem-Lab/2024_DKU_OS/issues)

Thank you