2025 Operating System

Lab 1. CPU Virtualization

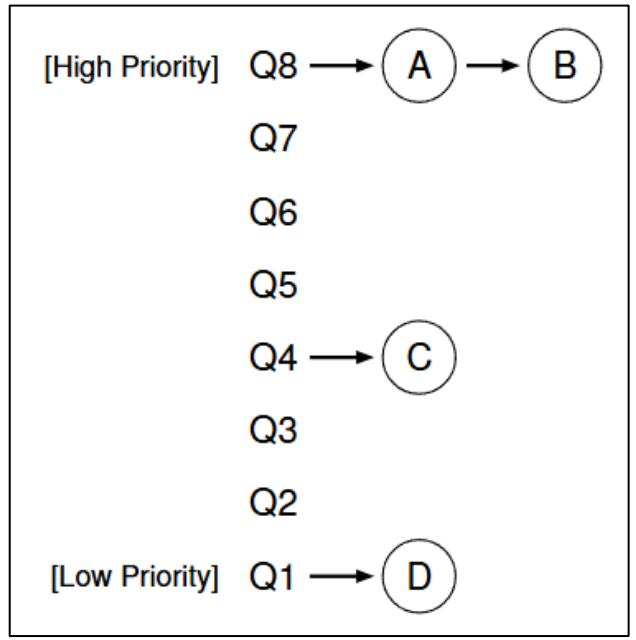
2025.04.01

T.A. 오여진

yeojinoh@dankook.ac.kr

Lab 1: CPU Virtualization

- Scheduling
  - 아래 4가지 Scheduler Simulator 구현
    - Round-Robin(RR)
    - Multi Level Feedback Queue(MLFQ)
    - Lottery
    - Stride



MLFQ Scheduling 예시

  - Context Switch Time, 워크로드에 따른 scheduling 결과 분석을 목표

Environment Setting

- https://github.com/DKU-EmbeddedSystem-Lab/2025_DKU_OS/
- Ubuntu 사용 (타 OS 사용 X)
- Lab0 매뉴얼을 보고 환경 설정 권장

- Lab 1 환경 구성
  - $ git clone https://github.com/DKU-EmbeddedSystem-Lab/2025_DKU_OS/
  - $ cd 2025_DKU_OS
  - $ sudo install_package.sh
    - 과제에 필요한 패키지 설치 (Vim, g++, OpenSSL, Gtest, …)
  - $ cd lab1
  - $ run.sh
    - 소스코드 빌드 및 실행

Workload Types

- Dynamic Workload – RR, MLFQ
  - Arrival Time: 프로세스가 ready-queue에 도착한 시간
  - Service Time: 프로세스가 완료될 때까지 걸리는 시간

Dynamic Workload A

Process Name	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

Dynamic Workload B

Process Name	Arrival Time	Service Time
1	0	10
2	1	3
3	2	2
4	6	5
5	15	12
6	25	2
7	27	4
8	33	3

Workload Types

- Batch Workload – Lottery, Stride
  - Tickets: 프로세스에게 할당된 티켓의 개수
  - Batch 워크로드이기 때문에 Arrival Time은 모두 0으로 동일

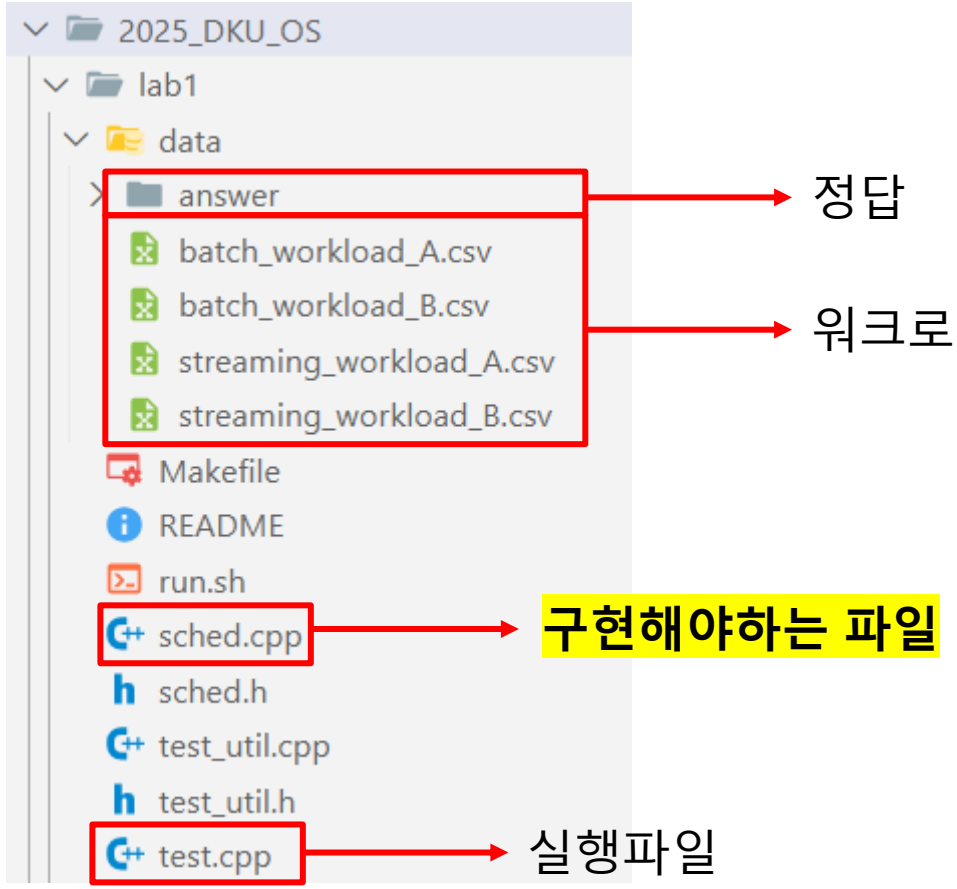
Batch Workload A

Process Name	Tickets	Service Time
1	50	4
2	100	8
3	150	6
4	200	2

Batch Workload B

Process Name	Tickets	Service Time
1	20	5
2	40	1
3	50	6
4	80	2
5	130	8
6	180	4
7	200	3
8	300	10

Code Flow



<test.cpp>: main 함수 실행

```cpp
int main() {
    ::testing::InitGoogleTest();
    return RUN_ALL_TESTS();
}
```

<test.cpp>: 워크로드 파일, context switch time별 테스트 케이스 생성

```cpp
// 각 7가지 TEST_P(스케줄링 기법)는 각각 4가지 경우(workload 2가지 x switch_time 2가지)를 TEST 함
INSTANTIATE_TEST_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", 0.05)
    // std::make_tuple("B", 0.2)
  )
);
```

주석처리를 통해 원하는
테스트 케이스만 확인 가능

<test_util.cpp>: 워크로드 파일 파싱해서 job_queue_와 job_list_에 삽입

```cpp
// TEST_P 실행 전, 실행 됨
void SchedulerTest::SetUp() {
  load_batch_workload();
  load_streaming_workload();
}
```

<test.cpp : 스케줄러 클래스 생성

```cpp
TEST_P(SchedulerTest, FCFS) {
  sched_ = new FCFS(job_queue_, switch_time_);
}
```

<test_util.cpp>: 각 스케줄러 실행, 결과 출력

```cpp
void SchedulerTest::TearDown() {
  run_sched(sched_);
  print_order();
  print_stat();
  check_answer(sched_->get_name(), "completion");
  check_answer(sched_->get_name(), "order");
  delete sched_;
}
```

Scheduler Class

<sched.h>: Job class

```
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
    int tickets = 0; // 티켓 수
};
```

<sched.h>: Scheduler class

```
class Scheduler {
    protected:
        // 스케줄러 이름
        std::string name;
        // workload 작업들이 이름 순으로 정렬된 큐
        std::queue<Job> job_queue_;
        // workload 작업들이 이름 순으로 정렬된 list
        std::list<Job> job_list_;
        // 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, waiting_time 또한 이를 기준으로 함.
        double current_time_ = 0;
        // 현재 작업 (처음에는 존재하지 않는 job(name=0)으로 초기화되어 있음)
        Job current_job_;
```

<sched.h>: FCFS 구현 예시

```
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;
        }
};
```

필수!
완료된 작업의 stat을 기록하는 작업
추가하지 않으면 test 정답처리 안됨

모든 작업이 완료된 경우, -1 반환해야함

sched.cpp (Implementation)

```
/*
*   DKU Operating System Lab
*       Lab1 (Scheduler Algorithm Simulator)
*       Student id :
*       Student name :
*       Date :
*       Contents :
*/
```

필수!
상단에 인적정보 기입

- RR, MLFQ
  - std::queue<Job> job_queue_ 사용

```
class RR : public Scheduler{
    private:
        int time_slice_;
        int left_slice_;
        std::queue<Job> waiting_queue;

    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;
        }

};
```

```
class FeedBack : public Scheduler {
private:
    std::queue<Job> queue[4]; // 각 요소가 하나의 큐인 배열 선언
    int quantum[4] = {1, 1, 1, 1};
    int left_slice_;
    int current_queue;

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
        */
        // Queue별 time quantum 설정
        if (name == "FeedBack_2i") {
            quantum[0] = 1;
            quantum[1] = 2;
            quantum[2] = 4;
            quantum[3] = 8;
        }
    }

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

};
```

sched.cpp (Implementation)

```
/*
*   DKU Operating System Lab
*       Lab1 (Scheduler Algorithm Simulator)
*       Student id :
*       Student name :
*       Date :
*       Contents :
*/
```

필수!
상단에 인적정보 기입

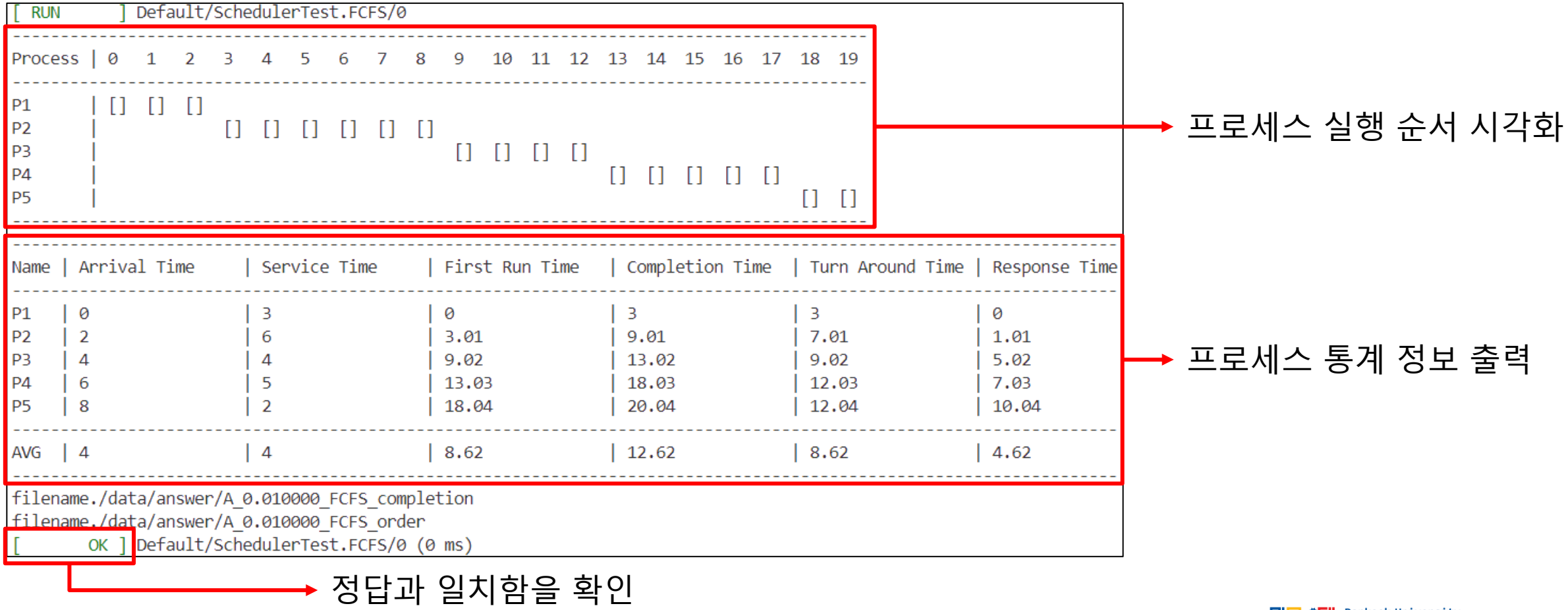
- Lottery, Stride
  - std::queue<Job> job_list_ 사용

```
class stride : public Scheduler{
    private:
        // 각 작업의 현재 pass 값과 stride 값을 관리하는 맵
        std::unordered_map<int, int> pass_map;
        std::unordered_map<int, int> stride_map;
        const int BIG_NUMBER = 10000; // stride 계산을 위한 상수 (보통 큰 수를 사용)

    public:
        stride(std::list<Job> jobs, double switch_overhead) : Scheduler(jobs, switch_overhead) {
            name = "stride";
                // job_list_에 있는 각 작업에 대해 stride와 초기 pass 값(0)을 설정
            for (auto &job : job_list_) {
                // stride = BIG_NUMBER / tickets (tickets는 0이 아님을 가정)
                stride_map[job.name] = BIG_NUMBER / job.tickets;
                pass_map[job.name] = 0;
            }
        }

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

```
class lottery : public Scheduler{
    private:
        int counter = 0;
        int total_tickets = 0;
        int winner = 0;
        std::mt19937 gen;  // 난수 생성기
        /*
        * 구현 (멤버 변수/함수 추가 및 삭제 가능)
        */

    public:
        lottery(std::list<Job> jobs, double switch_overhead) : Scheduler(jobs, switch_overhead) {
            name = "lottery";
            // 난수 생성기 초기화
            uint seed = 10; // 수정 금지
            gen = std::mt19937(seed);
            total_tickets = 0;
            for (const auto& job : job_list_) {
                total_tickets += job.tickets;
            }
        }

        int getRandomNumber(int min, int max) {
            std::uniform_int_distribution<int> dist(min, max);
            return dist(gen);
        }

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

Visualization

- 프로그램 실행 결과



```
[ RUN      ] Default/SchedulerTest.FCFS/0
-------------------------------------------------------------------------------------------
Process | 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19
-------------------------------------------------------------------------------------------
P1      | []  []  []
P2      |             []  []  []  []  []  []
P3      |                                 []  []  []  []
P4      |                                             []  []  []  []  []
P5      |                                                                 []  []
-------------------------------------------------------------------------------------------
```

프로세스 실행 순서 시각화

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	4.62

프로세스 통계 정보 출력

```
filename./data/answer/A_0.010000_FCFS_completion
filename./data/answer/A_0.010000_FCFS_order
[       OK ] Default/SchedulerTest.FCFS/0 (0 ms)
```

정답과 일치함을 확인

Project Requirements

- 요구사항
  - 스케줄링 알고리즘에 따라 프로세스를 정확한 순서대로 스케줄링하고, 모든 작업 정보를 정확히 기록하는 스케줄러를 작성하라
  - 스케줄러 종류는 다음과 같다 : RR, FeedBack, Lottery, Stride
  - 스케줄러는 생성자를 통해 실행할 작업들이 저장된 job_queue/job_list와 context switch time을 전달 받는다
  - 스케줄러는 run()함수가 호출될 때마다, 다음 1초간 실행할 작업 명을 반환한다
  - 스케줄러의 run()함수는 모든 작업이 완료된 경우, 그을 반환한다
  - 스케줄러는 job 구조체의 정보가 변경될 때마다, 이를 모두 update한다
  - 스케줄러는 완료된 job을 "end-jobs" 에 순서대로 저장 (push-back)한다

Workload Format

- 입력 (Workload)
  - 프로세스 (Job)의 개수 (n): 0 ≤ n ≤ 10 (개)
    - Job-queue에는 작업들이 arrival time순으로 정렬되어 저장되어 있다
    - Job-list에는 작업들이 ticket 개수 순으로 정렬되어 저장되어 있다
  - 각 프로세스 Service Time (s) : 1 ≤ 5 ≤ 100
  - 각 프로세스 Arrival Time (a) : 1 ≤ a ≤ 1,000
  - 문맥 교환 (Context Switch) Time (c) : 0.01 ≤ c ≤ 1
  - Test workload 수행 시, CPU는 작업이 완료될 때까지 idle한 경우가 존재하지 않는다

Implementation Details

- 구현해야 하는 내용
  - schd.cpp의 RR, FeedBack, Lottery, Stride 클래스를 구현한다
    - 각 클래스의 1) 생성자, 2) int run() 함수를 수정 및 작성하여 구현한다
      - 각 클래스의 위 2가시 함수의 선언은 수정할 수 없다
      - 생성자는 부모 생성자를 호출하고, name을 초기화 해야한다 (기존 내용을 수정하지 만 것)
    - 각 클래스의 멤버 변수/함수는 자유롭게 추가 가능하다
    - 반드시 C++로 구현하야 하고 라이브러리는 C++ STL만 사용 가능하다
    - sched.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

Code Submission

- 양식
  - 제목: os_lab1_학번_이름.cpp
    - Ex) os_lab1_32190617_김보승.cpp
  - 코드 상단에 작성자 정보 기입
  - 코드 설명하는 주석 달기 (line by line)
  - run.sh로 컴파일이 되고, 정상적으로 실행이 되어야 함
  - C++ 로 작성

Report Guidelines

- 보고서 내용
  - 구현한 소스코드 설명
  - Discussion
    - 여러 워크로드에서, 다양한 스케줄러 기법을 적용했을 경우, 서로 다른 지표에 의한 분석
      - ex) {Workload A}에서 Context Switch Time이 {0.01}일 때, {Turnaround Time}이 가장 {낮은} 정책은 무엇인가?
        이유를 스케줄링 규칙과 실험 결과를 바탕으로 설명하시오
    - 과제를 진행하며 새롭게 배운 점/ 어려웠던 점
    - 기타 내용 자유롭게 작성

Report Submission

- 양식
  - 제목: os_lab1_학번_이름.pdf
    - Ex) os_lab1_32190617_김보승.pdf
  - 코드 및 터미널 화면 첨부시 흰색 바탕으로 캡처
    - VS code 사용자: VS Code 테마 변경 방법
    - Linux 터미널: 리눅스 터미널 색상 변경 방법

Grading Criteria

- 구현
  - 실행 결과가 정답과 일치하는가? (모든 test 결과가 OK인지 확인)
  - 소스코드에 각 줄(or 코드 블록)마다 주석이 잘 작성되어 있는가?
  - 주어진 run.sh을 통해 컴파일 및 실행이 정상적으로 동작하는가?
    - 별도의 컴파일 방법 또는 실행 방법을 보고서에 서술해도 인정되지 않음
- 보고서
  - 각 스케줄링 기법의 성능을 그래프 등의 지표를 활용하여 비교/분석하였는가?
  - 구현한 소스코드에 대한 설명이 충분히 잘 작성되어 있는가?

Grading Criteria

- 총점 100점 = 구현 60점 + 보고서 30점 + 양식 10점

- 유의 사항
  - 지각 제출시, 하루에 10% 감점
  - 소스코드 제출 기준 미준수 시 → 구현 0점 + 형식 점수 0점
    - ex) 소스코드 텍스트로 제출, make/실행 안됨, …
  - 인적사항 주석 미작성, 파일명/형식 미 준수 시 → 형식 점수 0점

구분	세부사항	점수
구현	RR	15
	MLFQ	15
	Lottery	15
	Stride	15
보고서	구현 내용 설명	15
	Discussion	15
양식	주어진 양식 준수	10

Thank You

2025.04.01
T.A. 오여진
yeojinoh@dankook.ac.kr