1) 소스코드 설명

먼저 각각의 모든 Run()함수는 대체적으로 2개 혹은 3개의 부분으로 쪼갤 수 있음

2개: 선택, 실행직전(실행과 거의 동일)

3개: 실행 후 처리(혹은 실행 전 처리), 선택, 실행직전

A. 다이나믹 워크로드

i. RR 스케쥴러

클래스 맴버

```
class RR : public Scheduler{
    private:
        //현재 스케줄러의 변하지 않는 총 TQ
        int time_slice_;
        //TQ카운팅용
        int left_slice_;
        //현재 RR방식으로 스케쥴 해야하는 대상들이 모인 큐(즉 레디큐)
        std::queue<Job> waiting_queue;
```

설명은 주석으로 대체

함수부

생성자

```
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;
    }
```

주석대체

Run()

조건 : 반드시 다음 1초(1초 또는 1단위 작업시간)동안 스케쥴링 되어야할 작업의 이름을 반환

```
    //할당된 작업이 없고, => current_job_.name == 0
    // job queue가 비어있지 않은 경우 작업 시작
    //즉 맨처음 시작임
    if(current_job_.name == 0 && !job_queue_.empty()){
        current_job_ = job_queue_.front(); // 현재 작업을맨 앞작업으로 할당
        job_queue_.pop(); // 잡큐에서 팝으로 제거
    }
```

맨 처음 시작시, 최초로 작업을 선택하는 부분

```
//TQ단일 단위(1) 실행 완료시 항상, 새로이 입력된 작업이 존재하는지 확인
if(!job_queue_.empty())
{
    if(job_queue_.front().arrival_time <= current_time_){
        waiting_queue.push(job_queue_.front());
        job_queue_.pop();
    }
}
```

항상 단위작업이 끝난다면, 새로운 작업이 도착했는지 검사

의 까지가 어느정도는 실행 직후의 상황을 저장하는 단계

이후의 부분은 다음 작업을 선별하는 단계임

```
//작업 종료 검사,(작업이 TQ중간에 종료 되는지도 포함)
if(current_job_.remain_time == 0){ //현재 작업이 끝난 경우임임
    //완수 시간 저장
    current_job_.completion_time = current_time_;
    //완수 잡 벡터 저장
    end_jobs_.push_back(current_job_);
    //큐 검사
    //동시에 전부 빈경우
    if(waiting_queue.empty() && job_queue_.empty()){
        return -1;
    }

    //현재 작업 변경 및 대기큐 팝, 시간변경
    current_job_ = waiting_queue.front();
    waiting_queue.pop();
    current_time_ = current_time_ + switch_time_;
    //시간 변경에 따른 검사
    if(!job_queue_.empty())
    {
        if(job_queue_.front().arrival_time <= current_time_){
            waiting_queue.push(job_queue_.front());
            job_queue_.pop();
        }
    }
    //TQ카운터 초기화
    left_slice_ = time_slice_;

}else if(left_slice_ == 0){
```

검사 1단계 : 현재 작업의 완료 여부

해당 시: 현재 작업의 완료를 업데이트하고, end job에 업데이트 및 완전종료 검사.

재-스케쥴해야한다면, 조건에 따라 검사하고, 새로이 현재 작업 할당

검사 단계 2: 현재 작업이 주어진 TQ를 전부 소진하였는지 판단

```
}else if(left_slice_ == 0){
    //할당된 TQ전부사용
    //TQ전부 사용과, 작업 완전 종료는, 둘중 한놈만 검사하면 된다.
    //=> Q: 잡 완료됨?
    //  A1: ㅇㅇ => 어 그럼 교체해
    //  A2: ㄴㄴ => 어그럼 TQ는 전부씀?(지금 여기)
    //의미는 위에서 안걸러진
    //즉 아직 실행시간이 남아있다는 뜻임
    //현재 대기 큐가 비어있는지 확인
    if(waiting_queue.empty()){
        //TQ코인 충전
        left_slice_ = time_slice_;
        //이후 잡 교체 없음 => 문맥교환 x
    }else{;
        //현재 잡을 대기 큐에 삽입
        //새로운 잡을 먼저 삽입해줄 필요없음
        //=> 위에서 이미 기본단위(1)이 끝난 순간, 새로운 입력이 있는지 검사했음
        waiting_queue.push(current_job_);
        //현재 작업 교체
        current_job_ = waiting_queue.front();
        //대기큐 팝(즉 ready state => run state로 돌입)
        waiting_queue.pop();
        //context swtich time 추가
        current_time_ = current_time_ + switch_time_;
        //시간 변경에 따른 검사
        if(!job_queue_.empty())
        {
            if(job_queue_.front().arrival_time <= current_time_){
                waiting_queue.push(job_queue_.front());
                job_queue_.pop();
            }
        }
        //TQ코인 충전
        left_slice_ = time_slice_;
    }
}
```

TQ소진시 반드시 점검을 해야하므로, 주어진 조건에 따라서 검사 후 새로운 작업 할당의 부분까지가 조건검사부분이고, 이후는 실행 직전 부분임

아래는 실행 직전 부임:

실행직전 1: 첫실행 검사

```
//첫 실행 업데이트
if(current_job_.service_time == current_job_.remain_time){

    current_job_.first_run_time = current_time_;
}
```

해당 부분을 조건검사가 아닌 첫실행 검사에 넣은 이유는

모든 조건을 검사하고, 스케쥴 직전에 검사하는 것이 보다

실행직전 2: 각종 데이터 업데이트

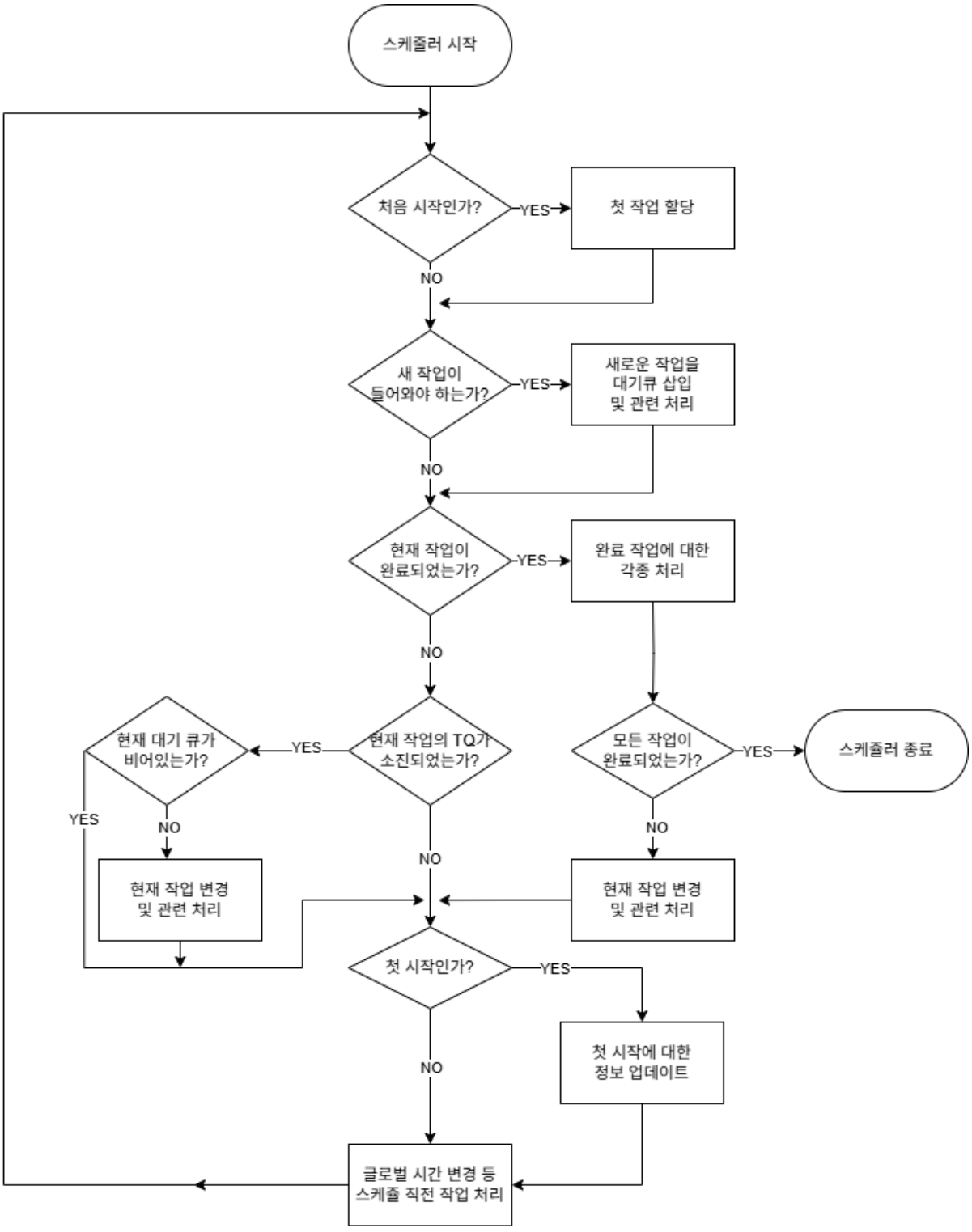
```
}
//위에서 쓴 조건에 따라서, 이거는 이후 1초동안 실행된 결과를 미리 연산하는 것

//글로벌타임 ++ => 다음 1초의 작업 이후 글로벌타임을 의미
current_time_++;
//현재 작업 남은시간 -- => 다음 1초의 작업 이후 잔여 시간을 의미
current_job_.remain_time--;
//TQ 조건 갱신 => 다음 1초의 작업 이후, TQ코인 개수 의미
left_slice_--;
return current_job_.name;
```

이후 첫부분부터 반복실행

아래는 간략한 순서도

매우 간략한 순서도 이기 때문에, 통상적인 순서도의 규칙을 준수하지 않음



ii. MLFQ 스케줄러

스케쥴러 맴버

```
private:
    std::queue<Job> queue[4];
    // 각 요소가 하나의 큐인 배열 선언
    //즉 위의 요놈은, 레디큐임
    int quantum[4] = {1, 1, 1, 1};
    //위의 요놈은 각 레벨의 큐별 TQ를 상징
    int left_slice_;
    //TQ카운터
    int current_queue;
    //재 스케쥴시, 처리를 위한 임시 큐 레벨
    int queue_level_for_re_sched;
```

함수부

생성자

```
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;
        }

        /*
        생성자 해석
        => is_2i에 걸리면, 아마 레디큐의 TQ가 2의 제곱으로 커지고
        아니면 그냥 RR마냥 정해진 레벨로 돌아가는듯?(아마도 초기화에 의하면 1)
        */
    }
    // 비어있지 않은, 가장 높은 우선순위의 레디큐 레벨 반환
    // 만약 전부 비어있다면 -1 반환
```

```
}
// 비어있지 않은, 가장 높은 우선순위의 레디큐 레벨 반환
// 만약 전부 비어있다면 -1 반환
int returnHighestQueueLevel(){
    for(int i =0; i <4; i++){
        if(!queue[i].empty()){
            return i;
        }
    }
    return -1;
}
int run() override {
```

run()

선택 이전 부분

```
int run() override {

    //할당된 작업이 없고, => current_job_.name == 0
    // job queue가 비어있지 않은 경우 작업 시작
    //즉 맨처음 시작임
    if(current_job_.name == 0 && !job_queue_.empty()){
        current_job_ = job_queue_.front(); // 현재 작업을맨 앞작업으로 할당
        job_queue_.pop(); // 잡큐에서 팝으로 제거
        // 첫 시작이므로 레벨 설정 필요, level zero queue
        current_queue = 0;
        //TQ를 현재 레벨의 큐에 맞게 설정
        left_slice_ = quantum[current_queue];
    }

    //단일 단위 작업 실행 이후, 항상 새로이 입력된 작업이 존재한느지 확인
    if(!job_queue_.empty()){
        if(job_queue_.front().arrival_time <= current_time_){
            queue[0].push(job_queue_.front());
            job_queue_.pop();
        }
    }
```

첫 작업을 할당하고, 새 작업을 검사

조건 검사 및 선택 부분

```
if(current_job_.remain_time == 0){
    //현재 작업이, 완료된 경우
    //완료시각 기록
    current_job_.completion_time = current_time_;
    //완료 큐 업데이트
    end_jobs_.push_back(current_job_);
    //모든 스케쥴 종료 검사
    if(returnHighestQueueLevel() == -1 && job_queue_.empty()){
        return -1;
    }
    //재스케쥴 필요
    //대기중 요소가 존재하는 가장 높은 레디큐 레벨 획득
    queue_level_for_re_sched = returnHighestQueueLevel();

    //레디큐가 전부 비어있을 경우
    if(queue_level_for_re_sched == -1){
        // printf("\nempty queue error\n");
        //말이 안되는 조건, 따라서 오류임
        return -1;
    }

    //현재 상태는 막 작업이 완료된 상태임

    //현재 작업 교체, 가장 최상위 대기큐의 잡을 배치
    current_job_ = queue[queue_level_for_re_sched].front();
    //방금 뽑은 잡, 해당 큐에서 제거
    queue[queue_level_for_re_sched].pop();
    //문맥교환 context swtich 시간 연산
    current_time_ = current_time_ + switch_time_;
    //현재 방금 뽑은 큐의 레벨을 기억
    current_queue = queue_level_for_re_sched;
    //방금 뽑은 큐의 레벨에 맞는, TQ코인 할당
    left_slice_ =  quantum[current_queue];
    //교환에 따른 새로운 작업 검사
    if(!job_queue_.empty()){
        if(job_queue_.front().arrival_time <= current_time_){
            queue[0].push(job_queue_.front());
            job_queue_.pop();
        }
    }
} else if(left_slice_ == 0){
```

현재 선택 되어있는 작업이, 완료되었는지 검사 및 선택

```
} else if(left_slice_ == 0){

    //현재 할당받은 TQ를 모두 소진시
    //즉, 방금전 작업이 완료된것이 아님
    ////////////큐변환

    if(!job_queue_.empty()){
        if(job_queue_.front().arrival_time <= current_time_){
            queue[0].push(job_queue_.front());
            job_queue_.pop();
        }
    }
    //할당할 작업이 있는 큐 레벨 획득
    queue_level_for_re_sched = returnHighestQueueLevel();
```

```
queue_level_for_re_sched = returnHighestQueueLevel();

//대기 큐가 비어있는 경우
if(queue_level_for_re_sched == -1 ){
    //이게 아래의 큐 하향조정 작업보다 항상 우선되어야함
    //고대로 다시 해당 레벨에 맞는 TQ할당
    left_slice_ = quantum[current_queue];
}else if(queue_level_for_re_sched > current_queue + 1){
    ///지금 작업에 대한 큐를 하향조정해도, 현재 대기큐에 있는 가장 최상-우선순위 작업보다 우선되어야함
    //즉 그냥 가상으로 큐만 내리고, 따로 입출력 xx
    //그냥 다시 스케쥴
    //현재 작업에 대한 큐를 한칸 아래로 조정
    //즉 레디큐에 넣었다 빼는 과정만 제거하고, 마치 그렇게 동작한것처럼 설정
    current_queue = current_queue + 1;
    //해당 레디큐 레벨에 맞는 TQ설정
    left_slice_ = quantum[current_queue];
}else{
    //아무 조건도 필요없고, 현재 작업을 큐레벨 하향하고, 새로운 최상위 작업을 할당받음
    //현재 작업 하향조정
    if(current_queue == 3){
        //큐 레벨은 최대 3(4단계)까지이므로 넘어가지 않게 처리
        queue[3].push(current_job_);
    }else{
        // 아무 조건 필요없이, 현재 동작했던 잡의 큐레벨을 하나 하향
        queue[current_queue + 1].push(current_job_);
    }
    //새 최고-우선순위 잡 할당
    current_job_ = queue[queue_level_for_re_sched].front();
    //방금 할당한 잡, 큐에서 제거
    queue[queue_level_for_re_sched].pop();
    //교환시간추가
    current_time_ = current_time_ + switch_time_;
    //현재 큐 레벨 갱신
    current_queue = queue_level_for_re_sched;
    //현재 큐 레벨에 맞는 TQ 할당
    left_slice_ = quantum[queue_level_for_re_sched];
    //시간흐름에 따른 새 잡 검색
    if(!job_queue_.empty()){
        if(job_queue_.front().arrival_time <= current_time_){
            queue[0].push(job_queue_.front());
            job_queue_.pop();
        }
    }
```

현재 작업이 주어진 TQ 를 전부 소진하였는지 검사 및 선택

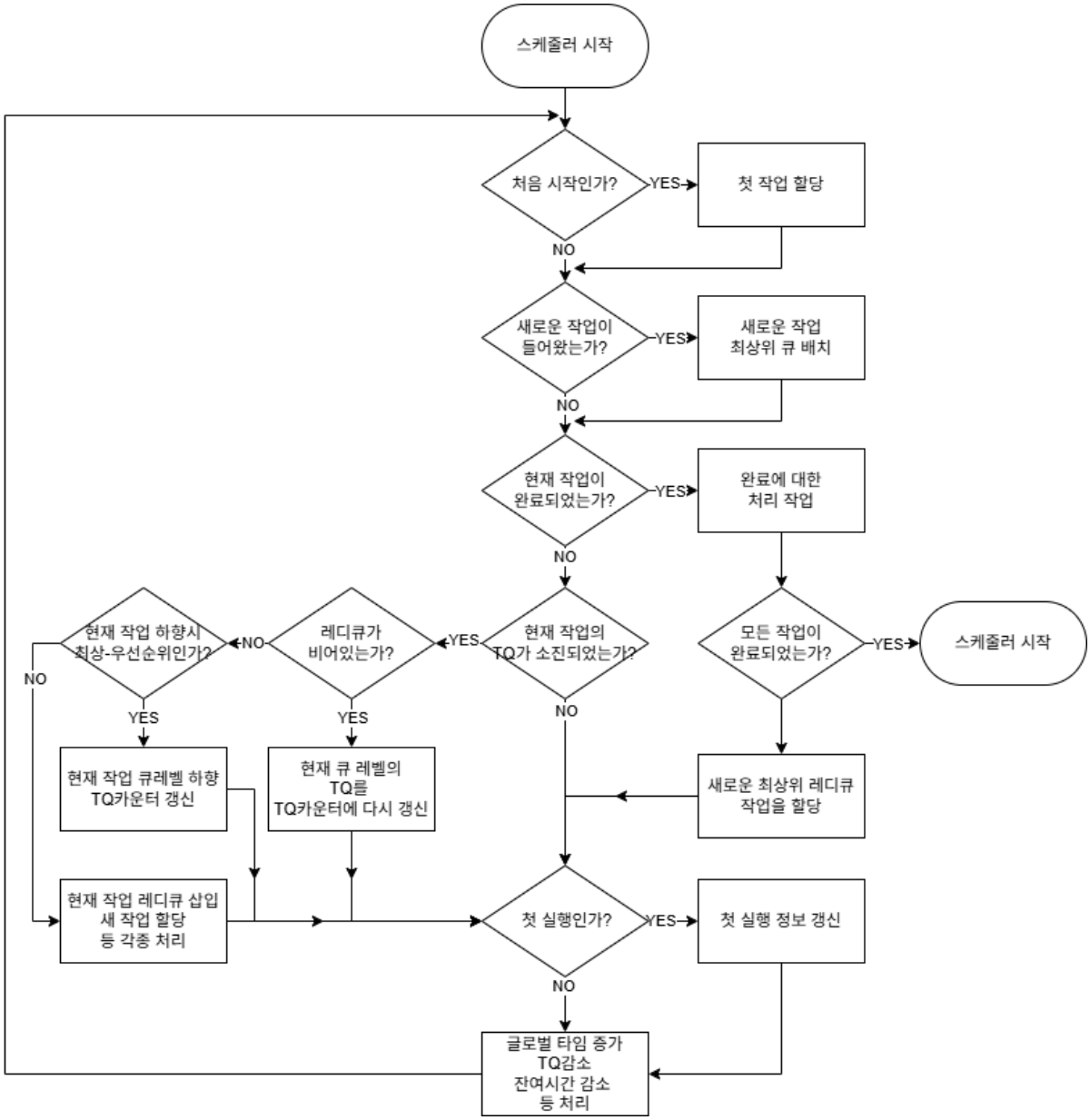
실행직전부분

```
    }
}
//TQ가 남아있다면, 즉 공통사항
//교체되었던 아니던 무조건 실행되어야하는 부분분

//첫실행 정보 갱신
if(current_job_.remain_time == current_job_.service_time){
    current_job_.first_run_time = current_time_;
}
//글로벌 타임 증가
current_time_++;
//현재 잡의 남은시간 감소
current_job_.remain_time--;
//현재 잡의 TQ를 감소
left_slice_--;
return current_job_.name;
```

실행직전 각종 필요 정보 검사 및 업데이트

아래는 간략하게 표현한 순서도



B. 배치 워크로드

i. LOTTERY 스케줄러

맴버 부분

```
ass Lottery : public Scheduler{
  private:
      int counter = 0;
      int total_tickets = 0;
      int winner = 0;
      std::mt19937 gen;  // 난수 생성기


      //아래 셋은 구현상 필요
      //job list직접 참조를 위한 잡-구조체형 포인터 변수
      Job* jobptr = NULL;
      //job list에서 추출 또는 뭐시깽이든 할때 필요한 임시 job name
      int target_job_name = 0;
      //가장 최근 실행된 job name 기억용
      int recent_job_name = 0;
      //최대 티켓값
      int max_ticket = 0;
      //최소 티켓값(고정)
      const int min_ticket = 1;
```

함수부분

생성자

```
public:
    Lottery(std::list<Job> jobs, double switch_overhead) : Scheduler(jobs, switch_overhead) {
        name = "Lottery";
        // 난수 생성기 초기화
        uint seed = 10; // seed 값 수정 금지
        gen = std::mt19937(seed);
        total_tickets = 0;
        for (const auto& job : job_list_) {
            total_tickets += job.tickets;
        }
    }
    //아마도 대충 min max 포함하는 범위에서 무작위변수 리턴하는 함수로 추정
    //사용시 1부터 티켓값까지로 사용(양끝단 포함)
```

난수 생성 함수

```
    }
    //아마도 대충 min max 포함하는 범위에서 무작위변수 리턴하는 함수로 추정
    //사용시 1부터 티켓값까지로 사용(양끝단 포함)
    int getRandomNumber(int min, int max) {
        std::uniform_int_distribution<int> dist(min, max);
        return dist(gen);
    }
```

최대 티켓을 업데이트하는 함수

```
    //최대 티켓 개수 갱신 함수
    void reCalTotalTicket(){
        total_tickets = 0;
        //각 joblist를 순회하면서, 모든 티켓값을 갱신
        for(const auto &job : job_list_){
            total_tickets += job.tickets;
        }
        //사용할 변수도 다시 갱신
        max_ticket = total_tickets;
    }
```

현재 뽑기에서 선택된 작업의 이름을 알아내는 함수

```
//RV에 각 잡의 티켓값을 빼가면서, 0포함 음수가 나오면 해당 구간인것으로판단.
int returnTargetJobName(){
    //하기전, 최대티켓 갱신
    reCalTotalTicket();
    //최소는 항상 1임; 랜덤값 추출
    int random_value = getRandomNumber(min_ticket, max_ticket);
    //리턴값 설정, 목표 잡 이름
    int target_job_name = -1;
    for(auto &job : job_list_){
        //job List를 순회하면서, 순차적으로(순서대로 순회함(잡0부터 끝까지<맨 처음 기준>))
        //무작위 난수에 대한 연산
        random_value = random_value - job.tickets;
        //만일 조건을 충족하면
        if(random_value <= 0){
            //반환 값 조정
            target_job_name = job.name;
            return target_job_name;
        }
    }
    //반환 , 해당 경우는 문제가 생긴 경우
    return target_job_name;
}
```

함수 이름을 바탕으로, joblist 에 접근 가능한 포인터를 반환하는 함수

```
//인자로 주어진 잡-이름을 바탕으로, job list에 접근하여, 동일한 이름을 가진 객체의 주소(*)를 반환
Job* returnJobPointer(int input_job_name){
    //반환용 포인터
    Job* jp = NULL;
    //job list를 순회
    for(auto &job: job_list_){
        //만일 내가 찾는 이름과 같은 이름을 가진 잡이 joblist에 존재시
        if(job.name == input_job_name){
            //조건에 맞는 job 주소 저장
            jp = &job;
            //해당 주소 반환
            return jp;
        }
    }
    //찾지 못한 경우(문제발생)
    return jp;
}
```

run()

선택 이전 부분

```
//현재 작업 이름이 0 => 첫실행
if(current_job_.name == 0){
    //실행해야할 작업의 이름을 획득
    target_job_name = returnTargetJobName();
    //해당 작업과 일치하는, joblist의 작업 정보 획득
    jobptr = returnJobPointer(target_job_name);
    //예외처리
    if(jobptr == NULL){
        return -1;
    }
    //현재 작업을, 위에서 얻은 작업 정보로 **(값)복사
    current_job_ = *jobptr;
    //최근 실행 작업이름을 갱신
    recent_job_name = current_job_.name;
    //현재 작업의 잔여시간 감소
    current_job_.remain_time--;
    //글로벌 시간 증가
    current_time_++;
    //반환
    //첫 시도이기에 로터리는 이러한 처리가 필요
    return current_job_.name;
}
```

첫 실행 검사

LOTTERY 의 경우, 맨 처음 첫 실행 당시에도, 무작위 추첨을 통한 예외를 처리해주어야함

```
}
//항상 단위작업을 이행 한 뒤, 현재 작업의 정보를 joblist의 일치하는 요소에 갱신
*jobptr = current_job_;
//마일 현재 작업이 완료되경우
```

항상 단위 작업이 끝나면, 역참조를 통해서, joblist 에 직접 데이터 갱신위한 부분

조건 검사 및 선택 부분

```
//현재 잡이 작업이 완료인지?
if(current_job_.remain_time == 0){
    //완료시간갱신
    current_job_.completion_time = current_time_;
    //해당 작업 다시 갱신
    *jobptr = current_job_;
    //완료된 작업, 완료벡터에 추가
    end_jobs_.push_back(*jobptr);

    //현재 잡 지우기
    //.earse()함수가, 이터레이터만 인자로 받기 때문에 아래와 같은 형식을 사용
    for(auto remove_target = job_list_.begin(); remove_target != job_list_.end(); ++remove_target){
        //조건에 맞는(지워야하는 작업) 작업 발견시시
        if(remove_target -> name == current_job_.name){
            //job list에서 제거거
            job_list_.erase(remove_target);
            break; // ***************매우중요, 지우고 탈출안하면 고장남(지워진 메모리 참조)
        }
    }
    //모든 작업 종료 검사
    if(job_list_.empty()){
        return -1;
    }

    //새작업 할당
    //새작업 이름 획득
    target_job_name = returnTargetJobName();
    //예외처리
    if(target_job_name == -1){
        return -1;
    }
    //이름에 알맞는 joblist의 작업 정보(주소) 획득
    jobptr =returnJobPointer(target_job_name);
    //예외처리
    if(jobptr == NULL){
        return -1;
    }
    //현재 작업 정보 갱신
    current_job_ = *jobptr;
    //글로벌 시간 갱신
    current time  = current time  + switch time ;
```

현재 작업이 완료된 경우

```
    current_time_ = current_time_ + switch_time_;
}else{
    //방금 스케쥴링으로 인해, 작업이 완료된것이 아님
    //따라서 그냥 교체만 필요
    //다음 작업 이름 획득
    target_job_name = returnTargetJobName();
    //방금 바로전에 실행되었던 작업과, 현재 선택된 작업이 동일한 경우
    if(recent_job_name == target_job_name){
        //아무 변경점도 필요 없음
    }else{
        //다른 작업의 경우
        //이름을 바탕으로 주소 획득
        jobptr = returnJobPointer(target_job_name);
        //예외처리
        if(jobptr == NULL){
            return -1;
        }
        //현재 작업 갱신
        current_job_= *jobptr;
        //글로벌 시간 갱신(문맥교환 시간)
        current_time_ = current_time_ + switch_time_;
    }
}
```

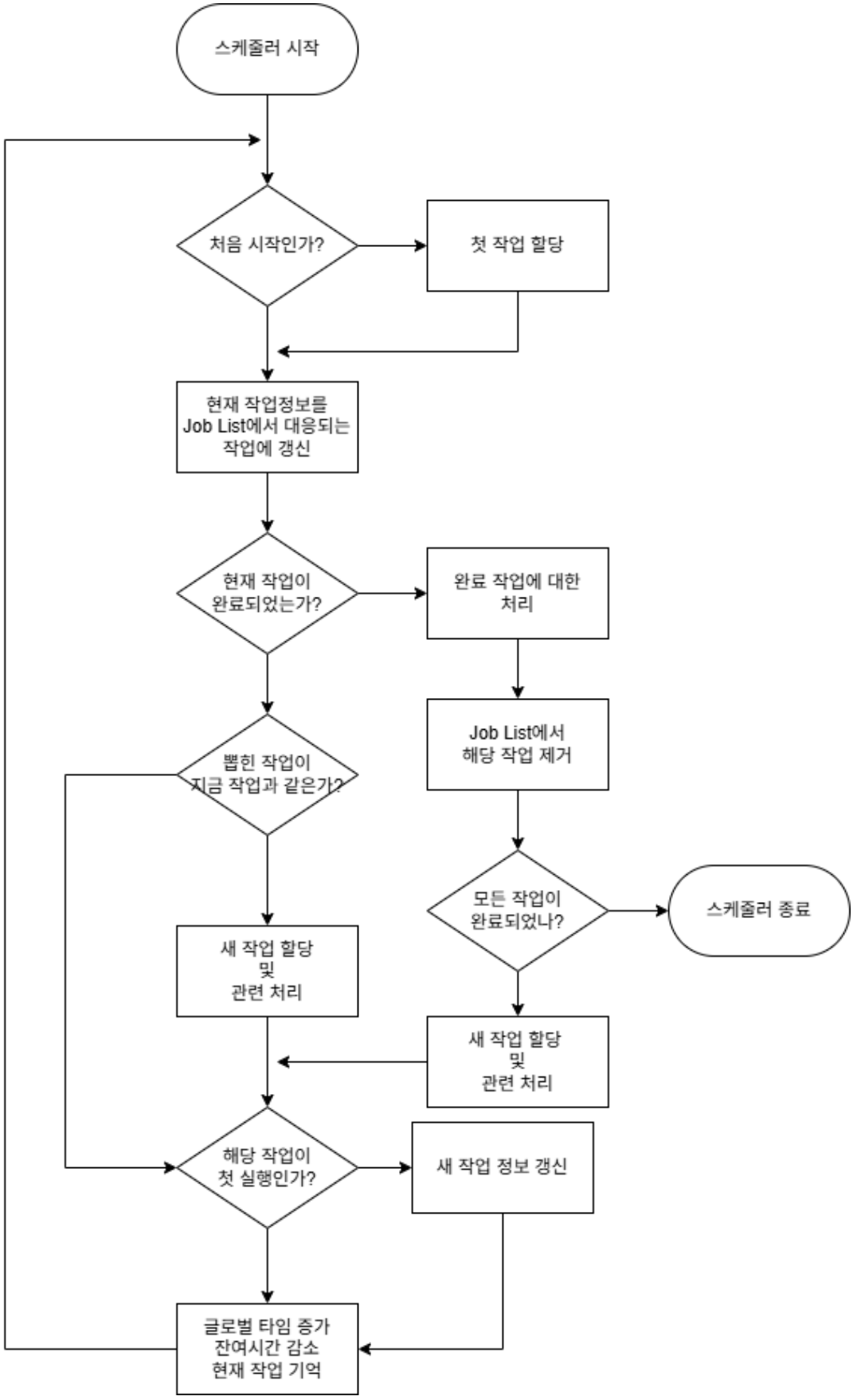
작업이 완료되지 않은 부분(교체 반드시 필요)

실행 직전 부분

```
//첫 실행 정보 갱신
if(current_job_.remain_time == current_job_.service_time){
    current_job_.first_run_time = current_time_;
}
// 현재 작업의 잔여시간 감소
current_job_.remain_time--;
//글로벌 시간 증가
current_time_++;
//가장 최신에 작동한 작업의 이름을, 현재 작업이름으로 갱신
recent_job_name = current_job_.name;
```

각종 검사 및 정보 업데이트

아래는 간략한 순서도


스케줄러 시작
처음 시작인가?
첫 작업 할당
현재 작업정보를
Job List에서 대응되는
작업에 갱신
현재 작업이
완료되었는가?
완료 작업에 대한
처리
Job List에서
해당 작업 제거
뽑힌 작업이
지금 작업과 같은가?
모든 작업이
완료되었나?
스케줄러 종료
새 작업 할당
및
관련 처리
새 작업 할당
및
관련 처리
해당 작업이
첫 실행인가?
새 작업 정보 갱신
글로벌 타임 증가
잔여시간 감소
현재 작업 기억

ii. STRIDE 스케줄러

클래스 맴버

```
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)


        //작업 교체등에 피요한 작업 이름
        int target_job_name = 0;
        //가장 최근에 작동한 작업 이름
        int recent_job_name = 0;
        //작업 교체 등에 필요한 잡-구조체 포인터 변수
        Job* jobptr = NULL;
```

함수부분

생성자

```
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;
    }
```

Pass 값 기준, 가장 작은 pass 값을 지닌 작업 이름 반환 함수

```
//현재 pass map을 기준으로, 가장 작은 value를 가진 pass맵의 key값(즉 잡-이름)을 반환
int returnMinJobName(){
    //정수 최댓값
    int min = 0x7fffffff;
    //반환할 작업 명
    int min_job_name = 0;
    //pass map을 순회
    for(auto &v : pass_map){
        //이때 unordered_map 이므로, 순서가 없음

        //따라서 원하는 결과, 순차적으로 검색한것과 동일한 효과
        //순차적으로 검색 했을 시, pass값이 동점인 경우, 앞의 작업을 우선해서 처리
        // => 라는 조건을 지키기 위해서 조건연산
        if(v.second < min){
            //패스값이 최소라 무조건 갱신
            min = v.second;
            min_job_name = v.first;
        }else if(v.second == min){
            //최솟값(패스 최소값)이 동일하면
            //더 앞선 작업명(앞의 작업)을 우선해서 갱신
            if(v.first < min_job_name){
                min_job_name = v.first;
                min = v.second;
            }
        }
    }
    //반환
    return min_job_name;
}
```

작업 이름을 바탕으로 JobList 접근 가능 포인터 반환 함수

```
//인자로 주어진 잡-이름을 바탕으로, job list에 접근하여, 동일한 이름을 가진 객체의 주소(*)를 반환
Job* returnJobPointer(int input_job_name){
    //반환 포인터 변수
    Job* jp = NULL;
    //잡리스트 순회
    for(auto &job: job_list_){
        //목표로 하는 잡 발견시
        if(job.name == input_job_name){
            //해당 잡 주소 저장
            jp = &job;
            //반환환
            return jp;
        }
    }
    // 이거 반환되면 뭔가 잘못된것
    return jp;
}
```

run()

선택 이전 부분

```
if(current_job_.name == 0){
    //stride에, 초기 값이 전부 0으로 초기화 되어있으므로, 1번 잡부터 시작
    if(!job_list_.empty()){
        //1번 작업 주소 할당
        jobptr = returnJobPointer(1);
        if(jobptr == NULL){
            return -1;
        }
    }
    //현재 잡 정보 할당
    //값복사임을 명심할 것
    current_job_ = *jobptr;
    //맨 처음 단계를 처리하기 위해서, 최근의 작업을 임의로 현재 잡으로 설정
    recent_job_name = current_job_.name;
}
```

```
//1초 단위 작업 후, 변경된 잡정보를 토대로, 잡 리스트의 매칭되는 객체를 갱신
//구현한 방법상, 반드시 처음에 job list객체를 역으로 갱신시켜주어야함
// std::cout << "AFTER SCHEDUL THE JOB LIST TARGET ELEMENT INFO UPDATE" << std::endl;
*jobptr = current_job_;
```

현재 작업의 정보를 대응되는 job List 의 요소에 업데이트

조건 검사 및 선택부분

```
//만일 현재 작업이 완료 된 경우
if(current_job_.remain_time == 0){
    //현재 작업 완료시간 갱신
    current_job_.completion_time = current_time_;
    //현재 작업을 잡리스트의 매칭되는 작업에 갱신
    *jobptr = current_job_;

    //잡 리스트에 접근해, 현재 실행중인 잡에 대한 완료 시간을 갱신
    jobptr -> completion_time = current_time_;

    //end job 벡터: 새로운 완료된 작업 추가
    end_jobs_.push_back(*jobptr);



    //다끝낸 잡을 리스트에서 제거, erase()때문에 해당 이터레이터 방식 사용
    for(auto remove_target = job_list_.begin(); remove_target != job_list_.end(); ++remove_target){
        //제거 잡 발견시
        if(remove_target -> name == current_job_.name){
            //제거거
            job_list_.erase(remove_target);
            break; // 탈출 *****(중요****)
        }
    }
    //다끝낸 잡에 대한 맵 제거, 패스, 스트라이드 둘다
    pass_map.erase(current_job_.name);
    stride_map.erase(current_job_.name);

    //모든 프로그램 완료 검사
    if(job_list_.empty()){
        return -1;
    }

    //다시 스케쥴링 해야하는 경우

    //가장 작은 pass값을 가진, 잡 이름 획득
    target_job_name = returnMinJobName();
    //잡이름을 바탕으로, job list의 객체에 직접 접근
    jobptr = returnJobPointer(target_job_name);
    //예외처리
    if(jobptr == NULL){
        return -1;
    }

    //위에서 접근한 객체를 복사하여, 현재 잡으로 할당
    current_job_ = *jobptr;
    //현재 시간을 문맥교환 시간을 더한 정보로 갱신
    current_time_ = current_time_ + switch_time_;
```

현재 작업이 완료된 경우에 대해 조건검사 및 선택

```
} else{
    //끝난거 아니면 무조건 재-스케쥴해야함(다시 돌려도 일단 찾아는 봐야함)

    //가장 작은 pass값을 가진, 잡 이름 획득
    target_job_name = returnMinJobName();

    //만일 지금 시도되어야하는 잡이, 직전에도 실행되었던 잡이라면
    if(target_job_name == recent_job_name){
        //그냥 넘어가기
        //맨 처음 케이스의 경우에도 여길 통해서 그냥 넘어감
    }else{
        //시도되어야하는 잡이, 이전 잡과 별개의 잡이라면
        //잡 이름을 바탕으로, 다시 job list의 매칭 객체에 접근
        jobptr = returnJobPointer(target_job_name);
        if(jobptr == NULL){
            return -1;
        }
        //접근한 객체를 복사하여, 현재 잡에 할당
        current_job_ = *jobptr;
        //문맥교환시간 추가
        current_time_ = current_time_ + switch_time_;
    }
}
```

완료 이외의 경우 조건 검사 및 처리 및 선택

실행 직전 부분

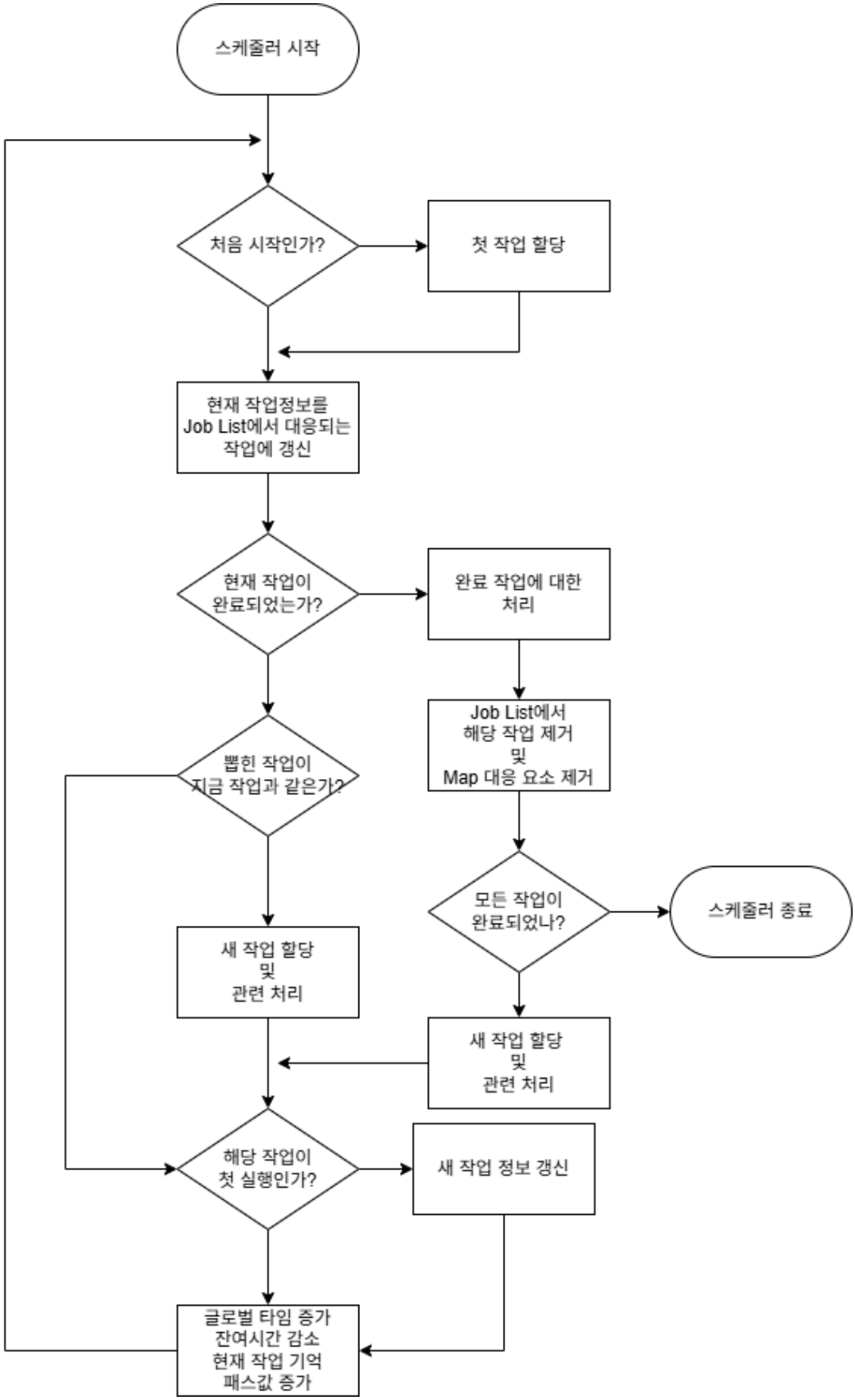
```
// 첫 스케쥴 일시, 정보 갱신
if(current_job_.remain_time == current_job_.service_time){
    current_job_.first_run_time = current_time_;
}
```

첫시작 정보 업데이트

```
    //글로벌 시간 증가
    current_time_++;
    //현재 잡의 남은시간 감소
    current_job_.remain_time--;
    //현재 잡에 해당하는 패스맵의 값 stride만큼 증가
    pass_map[current_job_.name] = pass_map[current_job_.name] + stride_map[current_job_.name];
    //실행될 잡의 이름을 임시 저장
    recent_job_name = current_job_.name;
    //다음 1초 뒤 스케쥴링될 작업명 반환
    return current_job_.name;
```

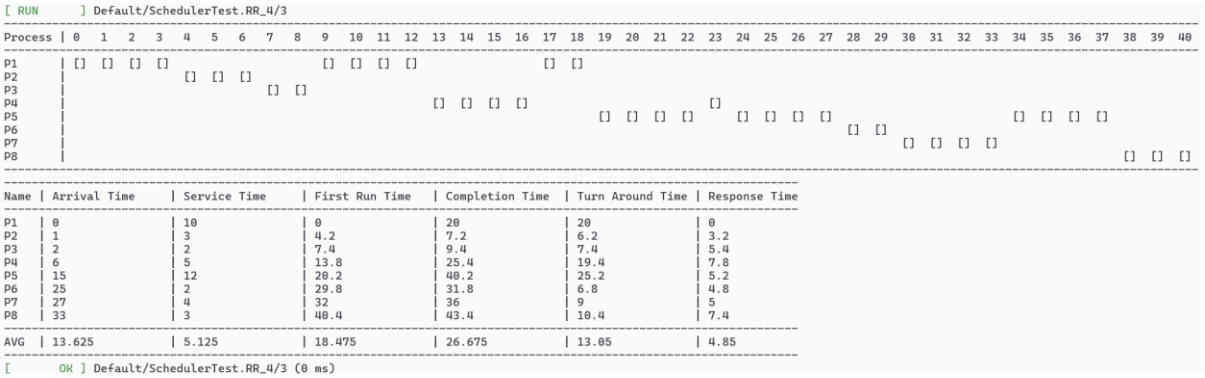
각종 필요 정보 업데이트

아래는 간략한 순서도


스케줄러 시작
처음 시작인가?
첫 작업 할당
현재 작업정보를
Job List에서 대응되는
작업에 갱신
현재 작업이
완료되었는가?
완료 작업에 대한
처리
Job List에서
해당 작업 제거
및
Map 대응 요소 제거
뽑힌 작업이
지금 작업과 같은가?
모든 작업이
완료되었나?
스케줄러 종료
새 작업 할당
및
관련 처리
새 작업 할당
및
관련 처리
해당 작업이
첫 실행인가?
새 작업 정보 갱신
글로벌 타임 증가
잔여시간 감소
현재 작업 기억
패스값 증가

순서도상 표현하지 아니한, 작업 선택, 작업 제거 등을 제외하면, lottery 와 stride 는 굉장히 유사한 모습을 보인다.

2) Discussion



```
[ RUN      ] Default/SchedulerTest.RR_4/3
Process | 0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
P1      | [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] [] []
P2      |                 [] [] []
P3      |                       [] []
P4      |                                   [] [] [] []                                                 []
P5      |                                                     [] [] [] []    [] [] [] []                                 [] [] [] []
P6      |                                                                                                    [] []
P7      |                                                                                                       [] [] [] []
P8      |                                                                                                                   [] [] []
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
P1   | 0                | 10               | 0                | 20               | 20               | 0
P2   | 1                | 3                | 4.2              | 7.2              | 6.2              | 3.2
P3   | 2                | 2                | 7.4              | 9.4              | 7.4              | 5.4
P4   | 6                | 5                | 13.8             | 25.4             | 19.4             | 7.8
P5   | 15               | 12               | 20.2             | 40.2             | 25.2             | 5.2
P6   | 25               | 2                | 29.8             | 31.8             | 6.8              | 4.8
P7   | 27               | 4                | 32               | 36               | 9                | 5
P8   | 33               | 3                | 40.4             | 43.4             | 10.4             | 7.4
AVG  | 13.625           | 5.125            | 18.475           | 26.675           | 13.05            | 4.85
[       OK ] Default/SchedulerTest.RR_4/3 (0 ms)
```

RR 방식에서 워크로드 B 를 기준으로 스케쥴 RR_4/3 에서

Q : P4 는 왜 서비스시간이 5 임에도, 다른 서비스시간이 10 단위에 해당하는 작업들처럼 굉장히 오래 걸렸나?

A :

RR 의 특성 때문이다.

RR 은 정해진 TQ 를 기준으로 작업을 할당하고, 동시에 해당 작업은 들어온다고 무조건적으로 우선실행이 아닌, Ready Queue 에 들어선 뒤, 차례가 오면 시작된다.

이때, P4 를 살펴보면 도착은 6 인데, 첫 시작은 13.8 에 시작되어 도착에 비해 처음 시작되는데 굉장히 오래 걸렸다는 것을 알 수 있다.

또한, TQ 자체가 4 로 보이기 때문에, 해당 기준에서 P4 는 서비스시간이 5 이다. 따라서 RR 기준으로는 P4 는 1 이라는 굉장히 적은 잔여시간을 가졌음에도, 다른 2 개의 작업이 우선 수행되기를 기다려야 했었다.

정리 : 따라서 P4 는 도착한 뒤 대기 시간도 길었으며, 또한 해당 RR 스케쥴러의 TQ 의 시간에 의해서, 굉장히 적은 잔여 작업시간을 가졌음에도 불구하고, 다른 작업들이 먼저 스케쥴 되는 것을 기다리고 있었기 때문에, turn around time 이 굉장히 크게 측정되었다.

결론 : RR 의 경우, 먼저 레디큐에 적재되어있는 작업이 많다면, 새로운 작업은 상당히 뒤에 시작될 가능성이 있고, 이것은 TQ 가 크면 클수록 더욱 심해질 수 있다. 또한, 스케쥴러 자체가 부여하는 TQ 가 스케쥴되는 작업들의 평균 처리시간과도 어느정도 상성이 맞아야 한다.
예를들어 대분의 작업이 (4 의 배수 +1)의 정도의 크기를 가진다면, 그리고 스케쥴러의 TQ 가 4 의 배수라면, 해당 잔여 시간 1 때문에 작업들은 완수되지 못하고, 다른 작업이 먼저 스케쥴 되기를 기다려야 할 것이다. 이 또한 작업들이 많아지면 많아질수록 심해질 것이다.

3) 힘들고 어려웠던점 및 배운 점

힘들었던 점

힘들었던 첫번째는, C++자체다.

여태껏 거의 C 로만 학습해와서, 비슷한 유형은 쉬웠으나(전체적인 생김새 등), 쓰지 않던 vector 나 list queue map 등은 써먹기가 조금 까다로워(값복사 vs 주소복사 등) 지속적으로 검색해가면서 사용하여, 조금은 힘들었던 것 같다.

두번째는 정해진 양식 준수이다.

개인적인 추측이지만, 아마도 그냥 단순하게 맨땅에서 구현하는 것이 조금은 수월했을지도 모른다는 생각이 들곤 했다(내 취향의 입력양식과 내 취향의 출력양식 등). 이미 주어진 정해진 양식(클래스 구조체 및 반환 값 등등)을 읽고 이해하여 그것에 맞게 동작하게 만드는 것이 제일 힘들었던 것 같다. 다행히도 FIFO 구조가 헤더파일에 예시로 구현되어 있어서 이해하는데 굉장한 도움이 되었지만, 만약 없었더라면 이해하는데 조금 더 많은 시간을 할애했을 듯하다.

배운 점

C++의 벡터나, list 등등

위에서 설명한 힘들었던 부분과 동일하다. 까다롭긴 했지만, 일부분에 한해서 해당 기능을 사용하는데 조금은 익숙해졌다.

스케쥴러가 동작하는 개략적인 이해

직접 구현을 하면서, 어느 부분에서 검사를해야하고, 어느 조건이 충족되어야하며, 어느 흐름으로 이어져 가는지는 파악했다.

코드 이해 능력의 중요성

위의 두번째 힘든점에서 이어지는 깨달음이다. 생각해보니 혼자 일할 것 아니면, 결국 남과 협업하고, 그 과정에서 남의 코드를 이해해야 할 일이 많고 혹은 어떠한 양식이나 규정을 지켜서 작업해야 할 일도 있을 것인데, 언제까지고 내 취향으로만 작업하는 것에는 한계가 있을 것 같다는 생각이 들었다. 아마도 교수님께서 강의시간 틈틈이 회사나 현장이야기를 하셔서, 이러한 생각까지 든 것이 아닌가 싶다.

```
        int run() override {


            //할당된 작업이 없고, => current_job_.name == 0
            // job queue가 비어있지 않은 경우 작업 시작
            //즉 맨처음 시작임
            if(current_job_.name == 0 && !job_queue_.empty()){
                current_job_ = job_queue_.front(); // 현재 작업을맨 앞작업으로
할당

                job_queue_.pop(); // 잡큐에서 팝으로 제거
            }
            //TQ 단일 단위(1) 실행 완료시 항상, 새로이 입력된 작업이 존재하는지 확인
            if(!job_queue_.empty())
            {
                if(job_queue_.front().arrival_time <= current_time_){
                    waiting_queue.push(job_queue_.front());
                    job_queue_.pop();
                }
            }
            //작업 종료 검사,(작업이 TQ 중간에 종료 되는지도 포함)
            if(current_job_.remain_time == 0){ //현재 작업이 끝난 경우임임
                //완수 시간 저장
                current_job_.completion_time = current_time_;
                //완수 잡 벡터 저장
                end_jobs_.push_back(current_job_);
                //큐 검사
                //동시에 전부 빈경우
                if(waiting_queue.empty() && job_queue_.empty()){
                    return -1;
                }

                //현재 작업 변경 및 대기큐 팝, 시간변경
                current_job_ = waiting_queue.front();
                waiting_queue.pop();
                current_time_ = current_time_ + switch_time_;
                //시간 변경에 따른 검사
                if(!job_queue_.empty())
                {
                    if(job_queue_.front().arrival_time <= current_time_){
                        waiting_queue.push(job_queue_.front());
                        job_queue_.pop();
                    }
                }
                //TQ 카운터 초기화
                left_slice_ = time_slice_;

            }else if(left_slice_ == 0){
                //할당된 TQ 전부사용
```

```
                //TQ 전부 사용과, 작업 완전 종료는, 둘중 한놈만 검사하면 된다.
                //=> Q: 잡 완료됨?
                //  A1: ㅇㅇ => 어 그럼 교체해
                //  A2: ㄴㄴ => 어그럼 TQ는 전부씀?(지금 여기)
                //의미는 위에서 안걸러진
                //즉 아직 실행시간이 남아있다는 뜻임
                //현재 대기 큐가 비어있는지 확인
                if(waiting_queue.empty()){
                    //TQ 코인 충전
                    left_slice_ = time_slice_;
                    //이후 잡 교체 없음 => 문맥교환 x
                }else{;
                    //현재 잡을 대기 큐에 삽입
                    //새로운 잡을 먼저 삽입해줄 필요없음
                    //=> 위에서 이미 기본단위(1)이 끝난 순간, 새로운 입력이 있는지
검사했음

                    waiting_queue.push(current_job_);
                    //현재 작업 교체
                    current_job_ = waiting_queue.front();
                    //대기큐 팝(즉 ready state => run state로 돌입)
                    waiting_queue.pop();
                    //context swtich time 추가
                    current_time_ = current_time_ + switch_time_;
                    //시간 변경에 따른 검사
                    if(!job_queue_.empty())
                    {
                        if(job_queue_.front().arrival_time <= current_time_){
                            waiting_queue.push(job_queue_.front());
                            job_queue_.pop();
                        }
                    }
                    //TQ 코인 충전
                    left_slice_ = time_slice_;
                }
            }

            //첫 실행 업데이트
            if(current_job_.service_time == current_job_.remain_time){

                current_job_.first_run_time = current_time_;
            }
            //위에서 쓴 조건에 따라서, 이거는 이후 1초동안 실행된 결과를 미리
연산하는 것

            //글로벌타임 ++ => 다음 1초의 작업 이후 글로벌타임을 의미
            current_time_++;
            //현재 작업 남은시간 -- => 다음 1초의 작업 이후 잔여 시간을 의미
            current_job_.remain_time--;
```

```
            //TQ 조건 갱신 => 다음 1초의 작업 이후, TQ코인 개수 의미
            left_slice_--;
            return current_job_.name;
        }

};

class FeedBack : public Scheduler {
    private:
        std::queue<Job> queue[4];
        // 각 요소가 하나의 큐인 배열 선언
        //즉 위의 요놈은, 레디큐임
        int quantum[4] = {1, 1, 1, 1};
        //위의 요놈은 각 레벨의 큐별 TQ를 상징
        int left_slice_;
        //TQ카운터
        int current_queue;
        //재 스케쥴시, 처리를 위한 임시 큐 레벨
        int queue_level_for_re_sched;
    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;
            }

            /*
            생성자 해석
            => is_2i에 걸리면, 아마 레디큐의 TQ가 2의 제곱으로 커지고
            아니면 그냥 RR마냥 정해진 레벨로 돌아가는듯?(아마도 초기화에 의하면 1)
            */
        }
        // 비어있지 않은, 가장 높은 우선순위의 레디큐 레벨 반환
        // 만약 전부 비어있다면 -1 반환
        int returnHighestQueueLevel(){
```

```
            for(int i =0; i <4; i++){
                if(!queue[i].empty()){
                    return i;
                }
            }
            return -1;
        }
        int run() override {

            //할당된 작업이 없고, => current_job_.name == 0
            // job queue가 비어있지 않은 경우 작업 시작
            //즉 맨처음 시작임
            if(current_job_.name == 0 && !job_queue_.empty()){
                current_job_ = job_queue_.front(); // 현재 작업을맨 앞작업으로
할당

                job_queue_.pop(); // 잡큐에서 팝으로 제거
                // 첫 시작이므로 레벨 설정 필요, level zero queue
                current_queue = 0;
                //TQ를 현재 레벨의 큐에 맞게 설정
                left_slice_ = quantum[current_queue];
            }

            //단일 단위 작업 실행 이후, 항상 새로이 입력된 작업이 존재한느지 확인
            if(!job_queue_.empty()){
                if(job_queue_.front().arrival_time <= current_time_){
                    queue[0].push(job_queue_.front());
                    job_queue_.pop();
                }
            }

            if(current_job_.remain_time == 0){
                //현재 작업이, 완료된 경우
                //완료시각 기록
                current_job_.completion_time = current_time_;
                //완료 큐 업데이트
                end_jobs_.push_back(current_job_);
                //모든 스케쥴 종료 검사
                if(returnHighestQueueLevel() == -1 && job_queue_.empty()){
                    return -1;
                }
                //재스케쥴 필요
                //대기중 요소가 존재하는 가장 높은 레디큐 레벨 획득
                queue_level_for_re_sched = returnHighestQueueLevel();

                //레디큐가 전부 비어있을 경우
                if(queue_level_for_re_sched == -1){
                    // printf("\nempty queue error\n");
                    //말이 안되는 조건, 따라서 오류임
```

```
                return -1;
            }

            //현재 상태는 막 작업이 완료된 상태임

            //현재 작업 교체, 가장 최상위 대기큐의 잡을 배치
            current_job_ = queue[queue_level_for_re_sched].front();
            //방금 뽑은 잡, 해당 큐에서 제거
            queue[queue_level_for_re_sched].pop();
            //문맥교환 context swtich 시간 연산
            current_time_ = current_time_ + switch_time_;
            //현재 방금 뽑은 큐의 레벨을 기억
            current_queue = queue_level_for_re_sched;
            //방금 뽑은 큐의 레벨에 맞는, TQ 코인 할당
            left_slice_ =  quantum[current_queue];
            //교환에 따른 새로운 작업 검사
            if(!job_queue_.empty()){
                if(job_queue_.front().arrival_time <= current_time_){
                    queue[0].push(job_queue_.front());
                    job_queue_.pop();
                }
            }
        } else if(left_slice_ == 0){

            //현재 할당받은 TQ를 모두 소진시
            //즉, 방금전 작업이 완료된것이 아님
            //////////////큐변환

            if(!job_queue_.empty()){
                if(job_queue_.front().arrival_time <= current_time_){
                    queue[0].push(job_queue_.front());
                    job_queue_.pop();
                }
            }
            //할당할 작업이 있는 큐 레벨 획득
            queue_level_for_re_sched = returnHighestQueueLevel();

            //대기 큐가 비어있는 경우
            if(queue_level_for_re_sched == -1 ){
                //이게 아래의 큐 하향조정 작업보다 항상 우선되어야함
                //고대로 다시 해당 레벨에 맞는 TQ 할당
                left_slice_ = quantum[current_queue];
            }else if(queue_level_for_re_sched > current_queue + 1){
                ///지금 작업에 대한 큐를 하향조정해도, 현재 대기큐에 있는 가장
최상-우선순위 작업보다 우선되어야함
                //즉 그냥 가상으로 큐만 내리고, 따로 입출력 xx
                //그냥 다시 스케쥴
                //현재 작업에 대한 큐를 한칸 아래로 조정
```

```
                    //즉 레디큐에 넣었다 빼는 과정만 제거하고, 마치 그렇게
동작한것처럼 설정
                    current_queue = current_queue + 1;
                    //해당 레디큐 레벨에 맞는 TQ 설정
                    left_slice_ = quantum[current_queue];
                }else{
                    //아무 조건도 필요없고, 현재 작업을 큐레벨 하향하고, 새로운
최상위 작업을 할당받음
                    //현재 작업 하향조정
                    if(current_queue == 3){
                        //큐 레벨은 최대 3(4 단계)까지이므로 넘어가지 않게 처리
                        queue[3].push(current_job_);
                    }else{
                        // 아무 조건 필요없이, 현재 동작했던 잡의 큐레벨을 하나 하향
                        queue[current_queue + 1].push(current_job_);
                    }
                    //새 최고-우선순위 잡 할당
                    current_job_ = queue[queue_level_for_re_sched].front();
                    //방금 할당한 잡, 큐에서 제거
                    queue[queue_level_for_re_sched].pop();
                    //교환시간추가
                    current_time_ = current_time_ + switch_time_;
                    //현재 큐 레벨 갱신
                    current_queue = queue_level_for_re_sched;
                    //현재 큐 레벨에 맞는 TQ 할당
                    left_slice_ = quantum[queue_level_for_re_sched];
                    //시간흐름에 따른 새 잡 검색
                    if(!job_queue_.empty()){
                        if(job_queue_.front().arrival_time <= current_time_){
                            queue[0].push(job_queue_.front());
                            job_queue_.pop();
                        }
                    }

                }
            }
            //TQ 가 남아있다면, 즉 공통사항
            //교체되었던 아니던 무조건 실행되어야하는 부분분

            //첫실행 정보 갱신
            if(current_job_.remain_time == current_job_.service_time){
                current_job_.first_run_time = current_time_;
            }
            //글로벌 타임 증가
            current_time_++;
            //현재 잡의 남은시간 감소
            current_job_.remain_time--;
            //현재 잡의 TQ 를 감소
```

```
            left_slice_--;
            return current_job_.name;
        }
};

class Lottery : public Scheduler{
    private:
        int counter = 0;
        int total_tickets = 0;
        int winner = 0;
        std::mt19937 gen;  // 난수 생성기


        //아래 셋은 구현상 필요
        //job list 직접 참조를 위한 잡-구조체형 포인터 변수
        Job* jobptr = NULL;
        //job list 에서 추출 또는 뭐시깽이든 할때 필요한 임시 job name
        int target_job_name = 0;
        //가장 최근 실행된 job name 기억용
        int recent_job_name = 0;
        //최대 티켓값
        int max_ticket = 0;
        //최소 티켓값(고정)
        const int min_ticket = 1;

    public:
        Lottery(std::list<Job> jobs, double switch_overhead) : Scheduler(jobs,
switch_overhead) {
            name = "Lottery";
            // 난수 생성기 초기화
            uint seed = 10; // seed 값 수정 금지
            gen = std::mt19937(seed);
            total_tickets = 0;
            for (const auto& job : job_list_) {
                total_tickets += job.tickets;
            }
        }
        //아마도 대충 min max 포함하는 범위에서 무작위변수 리턴하는 함수로 추정
        //사용시 1 부터 티켓값까지로 사용(양끝단 포함)
        int getRandomNumber(int min, int max) {
            std::uniform_int_distribution<int> dist(min, max);
            return dist(gen);
        }
        //최대 티켓 개수 갱신 함수
        void reCalTotalTicket(){
            total_tickets = 0;
            //각 joblist 를 순회하면서, 모든 티켓값을 갱신
            for(const auto &job : job_list_){
```

```
            total_tickets += job.tickets;
        }
        //사용할 변수도 다시 갱신
        max_ticket = total_tickets;
    }

    //RV에 각 잡의 티켓값을 빼가면서, 0포함 음수가 나오면 해당
구간인것으로판단.
    int returnTargetJobName(){
        //하기전, 최대티켓 갱신
        reCalTotalTicket();
        //최소는 항상 1임; 랜덤값 추출
        int random_value = getRandomNumber(min_ticket, max_ticket);
        //리턴값 설정, 목표 잡 이름
        int target_job_name = -1;
        for(auto &job : job_list_){
            //job List를 순회하면서, 순차적으로(순서대로 순회함(잡0부터
끝까지<맨 처음 기준>))
            //무작위 난수에 대한 연산
            random_value = random_value - job.tickets;
            //만일 조건을 충족하면
            if(random_value <= 0){
                //반환 값 조정
                target_job_name = job.name;
                return target_job_name;
            }
        }
        //반환 , 해당 경우는 문제가 생긴 경우
        return target_job_name;
    }
    //인자로 주어진 잡-이름을 바탕으로, job list에 접근하여, 동일한 이름을 가진
객체의 주소(*)를 반환
    Job* returnJobPointer(int input_job_name){
        //반환용 포인터
        Job* jp = NULL;
        //job list를 순회
        for(auto &job: job_list_){
            //만일 내가 찾는 이름과 같은 이름을 가진 잡이 joblist에 존재시
            if(job.name == input_job_name){
                //조건에 맞는 job 주소 저장
                jp = &job;
                //해당 주소 반환
                return jp;
            }
        }
        //찾지 못한 경우(문제발생)
        return jp;
    }
```

```
        int run() override {
            //현재 작업 이름이 0 => 첫실행
            if(current_job_.name == 0){
                //실행해야할 작업의 이름을 획득
                target_job_name = returnTargetJobName();
                //해당 작업과 일치하는, joblist의 작업 정보 획득
                jobptr = returnJobPointer(target_job_name);
                //예외처리
                if(jobptr == NULL){
                    return -1;
                }
                //현재 작업을, 위에서 얻은 작업 정보로 **(값)복사
                current_job_ = *jobptr;
                //최근 실행 작업이름을 갱신
                recent_job_name = current_job_.name;
                //현재 작업의 잔여시간 감소
                current_job_.remain_time--;
                //글로벌 시간 증가
                current_time_++;
                //반환
                //첫 시도이기에 로터리는 이러한 처리가 필요
                return current_job_.name;
            }
            //항상 단위작업을 이행 한 뒤, 현재 작업의 정보를 joblist의 일치하는
요소에 갱신
            *jobptr = current_job_;
            //만일 현재 작업이 완료된경우
            if(current_job_.remain_time == 0){
                //완료시간갱신
                current_job_.completion_time = current_time_;
                //해당 작업 다시 갱신
                *jobptr = current_job_;
                //완료된 작업, 완료벡터에 추가
                end_jobs_.push_back(*jobptr);

                //현재 잡 지우기
                //.earse()함수가, 이터레이터만 인자로 받기 때문에 아래와 같은 형식을
사용
                for(auto remove_target = job_list_.begin(); remove_target !=
job_list_.end(); ++remove_target){
                    //조건에 맞는(지워야하는 작업) 작업 발견시시
                    if(remove_target -> name == current_job_.name){
                        //job list에서 제거거
                        job_list_.erase(remove_target);
                        break; // ***************매우중요, 지우고 탈출안하면
고장남(지워진 메모리 참조)
                    }
```

```
    }
    //모든 작업 종료 검사
    if(job_list_.empty()){
        return -1;
    }

    //새작업 할당
    //새작업 이름 획득
    target_job_name = returnTargetJobName();
    //예외처리
    if(target_job_name == -1){
        return -1;
    }
    //이름에 알맞는 joblist 의 작업 정보(주소) 획득
    jobptr =returnJobPointer(target_job_name);
    //예외처리
    if(jobptr == NULL){
        return -1;
    }
    //현재 작업 정보 갱신
    current_job_ = *jobptr;
    //글로벌 시간 갱신
    current_time_ = current_time_ + switch_time_;
}else{
    //방금 스케쥴링으로 인해, 작업이 완료된것이 아님
    //따라서 그냥 교체만 필요
    //다음 작업 이름 획득
    target_job_name = returnTargetJobName();
    //방금 바로전에 실행되었던 작업과, 현재 선택된 작업이 동일한 경우
    if(recent_job_name == target_job_name){
        //아무 변경점도 필요 없음
    }else{
        //다른 작업의 경우
        //이름을 바탕으로 주소 획득
        jobptr = returnJobPointer(target_job_name);
        //예외처리
        if(jobptr == NULL){
            return -1;
        }
        //현재 작업 갱신
        current_job_= *jobptr;
        //글로벌 시간 갱신(문맥교환 시간)
        current_time_ = current_time_ + switch_time_;
    }
}
//첫 실행 정보 갱신
if(current_job_.remain_time == current_job_.service_time){
    current_job_.first_run_time = current_time_;
```

```
            }
            // 현재 작업의 잔여시간 감소
            current_job_.remain_time--;
            //글로벌 시간 증가
            current_time_++;
            //가장 최신에 작동한 작업의 이름을, 현재 작업이름으로 갱신
            recent_job_name = current_job_.name;

            return current_job_.name;
        }
};


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)


        //작업 교체등에 피요한 작업 이름
        int target_job_name = 0;
        //가장 최근에 작동한 작업 이름
        int recent_job_name = 0;
        //작업 교체 등에 필요한 잡-구조체 포인터 변수
        Job* jobptr = NULL;


    public:
        Stride(std::list<Job> jobs, double switch_overhead) : Scheduler(jobs,
switch_overhead) {
            name = "Stride";
                    // job_list_에 있는 각 작업에 대해 stride 와 초기 pass 값(0)을
설정
            for (auto &job : job_list_) {
                // stride = BIG_NUMBER / tickets (tickets 는 0 이 아님을 가정)
                stride_map[job.name] = BIG_NUMBER / job.tickets;
                pass_map[job.name] = 0;
            }
        }
        //현재 pass map 을 기준으로, 가장 작은 value 를 가진 pass 맵의 key 값(즉 잡-
이름)을 반환
        int returnMinJobName(){
            //정수 최댓값
            int min = 0x7fffffff;
            //반환할 작업 명
```

```
            int min_job_name = 0;
            //pass map을 순회
            for(auto &v : pass_map){
                //이때 unordered_map 이므로, 순서가 없음

                //따라서 원하는 결과, 순차적으로 검색한것과 동일한 효과
                //순차적으로 검색 했을 시, pass 값이 동점인 경우, 앞의 작업을
우선해서 처리
                // => 라는 조건을 지키기 위해서 조건연산
                if(v.second < min){
                    //패스값이 최소라 무조건 갱신
                    min = v.second;
                    min_job_name = v.first;
                }else if(v.second == min){
                    //최솟값(패스 최소값)이 동일하면
                    //더 앞선 작업명(앞의 작업)을 우선해서 갱신
                    if(v.first < min_job_name){
                        min_job_name = v.first;
                        min = v.second;
                    }
                }
            }
            //반환
            return min_job_name;
        }
        //인자로 주어진 잡-이름을 바탕으로, job list에 접근하여, 동일한 이름을 가진
객체의 주소(*)를 반환
        Job* returnJobPointer(int input_job_name){
            //반환 포인터 변수
            Job* jp = NULL;
            //잡리스트 순회
            for(auto &job: job_list_){
                //목표로 하는 잡 발견시
                if(job.name == input_job_name){
                    //해당 잡 주소 저장
                    jp = &job;
                    //반환환
                    return jp;
                }
            }
            // 이거 반환되면 뭔가 잘못된것
            return jp;
        }


        int run() override {
            if(current_job_.name == 0){
```

```
                //stride 에, 초기 값이 전부 0 으로 초기화 되어있으므로, 1 번 잡부터
시작
                if(!job_list_.empty()){
                    //1 번 작업 주소 할당
                    jobptr = returnJobPointer(1);
                    if(jobptr == NULL){
                        return -1;
                    }
                }
                //현재 잡 정보 할당
                //값복사임을 명심할 것
                current_job_ = *jobptr;
                //맨 처음 단계를 처리하기 위해서, 최근의 작업을 임의로 현재 잡으로
설정
                recent_job_name = current_job_.name;
            }

            //1 초 단위 작업 후, 변경된 잡정보를 토대로, 잡 리스트의 매칭되는 객체를
갱신
            //구현한 방법상, 반드시 처음에 job list 객체를 역으로 갱신시켜주어야함
            // std::cout << "AFTER SCHEDUL THE JOB LIST TARGET ELEMENT INFO
UPDATE" << std::endl;
            *jobptr = current_job_;


            //만일 현재 작업이 완료 된 경우
            if(current_job_.remain_time == 0){
                //현재 작업 완료시간 갱신
                current_job_.completion_time = current_time_;
                //현재 작업을 잡리스트의 매칭되는 작업에 갱신
                *jobptr = current_job_;

                //잡 리스트에 접근해, 현재 실행중인 잡에 대한 완료 시간을 갱신
                jobptr -> completion_time = current_time_;

                //end job 벡터: 새로운 완료된 작업 추가
                end_jobs_.push_back(*jobptr);



                //다끝낸 잡을 리스트에서 제거, erase()때문에 해당 이터레이터 방식
사용
                for(auto remove_target = job_list_.begin(); remove_target !=
job_list_.end(); ++remove_target){
                    //제거 잡 발견시
                    if(remove_target -> name == current_job_.name){
                        //제거거
                        job_list_.erase(remove_target);
```

```
                    break; // 탈출 *****(중요****)
                }
            }
            //다끝낸 잡에 대한 맵 제거, 패스, 스트라이드 둘다
            pass_map.erase(current_job_.name);
            stride_map.erase(current_job_.name);

            //모든 프로그램 완료 검사
            if(job_list_.empty()){
                return -1;
            }

            //다시 스케쥴링 해야하는 경우

            //가장 작은 pass 값을 가진, 잡 이름 획득
            target_job_name = returnMinJobName();
            //잡이름을 바탕으로, job list 의 객체에 직접 접근
            jobptr = returnJobPointer(target_job_name);
            //예외처리
            if(jobptr == NULL){
                return -1;
            }

            //위에서 접근한 객체를 복사하여, 현재 잡으로 할당
            current_job_ = *jobptr;
            //현재 시간을 문맥교환 시간을 더한 정보로 갱신
            current_time_ = current_time_ + switch_time_;
        } else{
            //끝난거 아니면 무조건 재-스케쥴해야함(다시 돌려도 일단 찾아는
봐야함)

            //가장 작은 pass 값을 가진, 잡 이름 획득
            target_job_name = returnMinJobName();

            //만일 지금 시도되어야하는 잡이, 직전에도 실행되었던 잡이라면
            if(target_job_name == recent_job_name){
                //그냥 넘어가기
                //맨 처음 케이스의 경우에도 여길 통해서 그냥 넘어감
            }else{
                //시도되어야하는 잡이, 이전 잡과 별개의 잡이라면
                //잡 이름을 바탕으로, 다시 job list 의 매칭 객체에 접근
                jobptr = returnJobPointer(target_job_name);
                if(jobptr == NULL){
                    return -1;
                }
                //접근한 객체를 복사하여, 현재 잡에 할당
                current_job_ = *jobptr;
                //문맥교환시간 추가
```

```
                current_time_ = current_time_ + switch_time_;
            }
        }

        // 첫 스케쥴 일시, 정보 갱신
        if(current_job_.remain_time == current_job_.service_time){
            current_job_.first_run_time = current_time_;
        }


        //글로벌 시간 증가
        current_time_++;
        //현재 잡의 남은시간 감소
        current_job_.remain_time--;
        //현재 잡에 해당하는 패스맵의 값 stride 만큼 증가
        pass_map[current_job_.name] = pass_map[current_job_.name] +
stride_map[current_job_.name];
        //실행될 잡의 이름을 임시 저장
        recent_job_name = current_job_.name;
        //다음 1 초 뒤 스케쥴링될 작업명 반환
        return current_job_.name;
    }
};
```

B. 실행 결과 전체

RR

```
[ RUN      ] Default/SchedulerTest.RR_1/0
-------------------------------------------------------------------------------
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      |                                         []              []
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
-------------------------------------------------------------------------------------------------------------
P1   | 0                | 3                | 0                | 4.02             | 4.02             | 0
P2   | 2                | 6                | 2.01             | 18.16            | 16.16            | 0.01
P3   | 4                | 4                | 5.04             | 17.15            | 13.15            | 1.04
P4   | 6                | 5                | 7.06             | 20.17            | 14.17            | 1.06
P5   | 8                | 2                | 10.09            | 15.13            | 7.13             | 2.09
-------------------------------------------------------------------------------------------------------------
AVG  | 4                | 4                | 4.84             | 14.926           | 10.926           | 0.84
-------------------------------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.RR_1/0 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.RR_1/1
```

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												[]			[]					

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	3	0	4.2	4.2	0
P2	2	6	2.1	19.6	17.6	0.1
P3	4	4	5.4	18.5	14.5	1.4
P4	6	5	7.6	21.7	15.7	1.6
P5	8	2	10.9	16.3	8.3	2.9
AVG	4	4	5.2	16.06	12.06	1.2

```
[       OK ] Default/SchedulerTest.RR_1/1 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.RR_1/2
```

Process	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
P1	[]		[]			[]					[]		[]		[]				[]		[]		[]																		
P2		[]			[]			[]																																		
P3				[]			[]																																		
P4									[]		[]		[]		[]					[]																					
P5																	[]		[]		[]		[]		[]		[]		[]												
P6																										[]		[]													
P7																													[]		[]		[]		[]						
P8																																				[]		[]		[]		

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	10	0	24.1	24.1	0
P2	1	3	1.05	8.35	7.35	0.05
P3	2	2	3.15	7.3	5.3	1.15
P4	6	5	8.4	18.85	12.85	2.4
P5	15	12	16.8	42.95	27.95	1.8
P6	25	2	25.2	28.3	3.3	0.2
P7	27	4	28.35	36.7	9.7	1.35
P8	33	3	34.65	40.9	7.9	1.65
AVG	13.625	5.125	14.7	25.9312	12.3063	1.075

```
[       OK ] Default/SchedulerTest.RR_1/2 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.RR_1/3
```

Process	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
P1	[]		[]			[]					[]		[]		[]		[]		[]		[]		[]																		
P2		[]			[]			[]																																		
P3				[]			[]																																		
P4									[]		[]		[]		[]		[]		[]																						
P5																					[]		[]		[]		[]		[]		[]		[]		[]		[]		[]	[]	[]	[]	[]
P6																										[]		[]													
P7																													[]		[]		[]		[]						
P8																																				[]		[]		[]		

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	10	0	29.8	29.8	0
P2	1	3	1.2	9.4	8.4	0.2
P3	2	2	3.6	8.2	6.2	1.6
P4	6	5	9.6	22.6	16.6	3.6
P5	15	12	16.8	48.4	33.4	1.8
P6	25	2	27.6	33.4	8.4	2.6
P7	27	4	30	41.8	14.8	3
P8	33	3	36	44.2	11.2	3
AVG	13.625	5.125	15.6	29.725	16.1	1.975

```
[       OK ] Default/SchedulerTest.RR_1/3 (0 ms)
[ RUN      ] Default/SchedulerTest.RR_4/0
```

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																		[]	[]	

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	17.04	15.04	1.01
P3	4	4	7.02	11.02	7.02	3.02
P4	6	5	11.03	20.06	14.06	5.03
P5	8	2	17.05	19.05	11.05	9.05
AVG	4	4	7.622	14.034	10.034	3.622

```
[       OK ] Default/SchedulerTest.RR_4/0 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.RR_4/1
-------------------------------------------------------------------------------------------
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      |                                                                 []  []
-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
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              | 17.4             | 15.4             | 1.1
P3   | 4                | 4                | 7.2              | 11.2             | 7.2              | 3.2
P4   | 6                | 5                | 11.3             | 20.6             | 14.6             | 5.3
P5   | 8                | 2                | 17.5             | 19.5             | 11.5             | 9.5
-------------------------------------------------------------------------------------------------------------
AVG  | 4                | 4                | 7.82             | 14.34            | 10.34            | 3.82
-------------------------------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.RR_4/1 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.RR_4/2
-----------------------------------------------------------------------------------------------------------------------------------------------------------
Process | 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
-----------------------------------------------------------------------------------------------------------------------------------------------------------
P1      | []  []  []  []                  []  []  []  []                  []  []
P2      |             []  []  []
P3      |                     []  []
P4      |                                 []  []  []  []                      []
P5      |                                                 []  []  []  []      []  []  []  []                      []  []  []  []
P6      |                                                                                     []  []
P7      |                                                                                         []  []  []  []
P8      |                                                                                                                     []  []  []
-----------------------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
-------------------------------------------------------------------------------------------------------------
P1   | 0                | 10               | 0                | 19.25            | 19.25            | 0
P2   | 1                | 3                | 4.05             | 7.05             | 6.05             | 3.05
P3   | 2                | 2                | 7.1              | 9.1              | 7.1              | 5.1
P4   | 6                | 5                | 13.2             | 24.35            | 18.35            | 7.2
P5   | 15               | 12               | 19.3             | 38.55            | 23.55            | 4.3
P6   | 25               | 2                | 28.45            | 30.45            | 5.45             | 3.45
P7   | 27               | 4                | 30.5             | 34.5             | 7.5              | 3.5
P8   | 33               | 3                | 38.6             | 41.6             | 8.6              | 5.6
-------------------------------------------------------------------------------------------------------------
AVG  | 13.625           | 5.125            | 17.65            | 25.6063          | 11.9813          | 4.025
-------------------------------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.RR_4/2 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.RR_4/3
-----------------------------------------------------------------------------------------------------------------------------------------------------------
Process | 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
-----------------------------------------------------------------------------------------------------------------------------------------------------------
P1      | []  []  []  []                  []  []  []  []                  []  []
P2      |             []  []  []
P3      |                     []  []
P4      |                                 []  []  []  []                      []
P5      |                                                 []  []  []  []      []  []  []  []                      []  []  []  []
P6      |                                                                                     []  []
P7      |                                                                                         []  []  []  []
P8      |                                                                                                                     []  []  []
-----------------------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
-------------------------------------------------------------------------------------------------------------
P1   | 0                | 10               | 0                | 20               | 20               | 0
P2   | 1                | 3                | 4.2              | 7.2              | 6.2              | 3.2
P3   | 2                | 2                | 7.4              | 9.4              | 7.4              | 5.4
P4   | 6                | 5                | 13.8             | 25.4             | 19.4             | 7.8
P5   | 15               | 12               | 20.2             | 40.2             | 25.2             | 5.2
P6   | 25               | 2                | 29.8             | 31.8             | 6.8              | 4.8
P7   | 27               | 4                | 32               | 36               | 9                | 5
P8   | 33               | 3                | 40.4             | 43.4             | 10.4             | 7.4
-------------------------------------------------------------------------------------------------------------
AVG  | 13.625           | 5.125            | 18.475           | 26.675           | 13.05            | 4.85
-------------------------------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.RR_4/3 (0 ms)
```

MLFQ

```
[ RUN      ] Default/SchedulerTest.FeedBack_1/0
-------------------------------------------------------------------------------------------
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      |                             []      []
-------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
-------------------------------------------------------------------------------------------------------------
P1   | 0                | 3                | 0                | 4.02             | 4.02             | 0
P2   | 2                | 6                | 2.01             | 20.18            | 18.18            | 0.01
P3   | 4                | 4                | 4.03             | 16.14            | 12.14            | 0.03
P4   | 6                | 5                | 6.05             | 19.17            | 13.17            | 0.05
P5   | 8                | 2                | 8.07             | 11.09            | 3.09             | 0.07
-------------------------------------------------------------------------------------------------------------
AVG  | 4                | 4                | 4.032            | 14.12            | 10.12            | 0.032
-------------------------------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.FeedBack_1/0 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.FeedBack_1/1
```

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									[]		[]									

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	3	0	4.2	4.2	0
P2	2	6	2.1	21.8	19.8	0.1
P3	4	4	4.3	17.4	13.4	0.3
P4	6	5	6.5	20.7	14.7	0.5
P5	8	2	8.7	11.9	3.9	0.7
AVG	4	4	4.32	15.2	11.2	0.32

```
[       OK ] Default/SchedulerTest.FeedBack_1/1 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.FeedBack_1/2
```

Process	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
P1	[]			[]					[]			[]		[]					[]		[]		[]							[]							[]				
P2		[]			[]					[]																																
P3			[]			[]																																			
P4					[]	[]			[]																																
P5																		[]	[]	[]		[]		[]															[]	[]	[]	[]	[]
P6																									[]	[]																
P7																											[]	[]	[]			[]										
P8																																	[]	[]	[]							

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	10	0	37.35	37.35	0
P2	1	3	1.05	10.4	9.4	0.05
P3	2	2	2.1	6.25	4.25	0.1
P4	6	5	6.3	15.65	9.65	0.3
P5	15	12	15.7	42.4	27.4	0.7
P6	25	2	25.05	27.05	2.05	0.05
P7	27	4	27.1	33.25	6.25	0.1
P8	33	3	33.3	36.3	3.3	0.3
AVG	13.625	5.125	13.825	26.0813	12.4563	0.2

```
[       OK ] Default/SchedulerTest.FeedBack_1/2 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.FeedBack_1/3
```

Process	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
P1	[]			[]					[]			[]				[]			[]		[]								[]							[]		[]			
P2		[]			[]					[]																																
P3			[]			[]																																			
P4					[]	[]			[]									[]																								
P5														[]	[]	[]			[]		[]								[]							[]			[]	[]	[]	[]
P6																								[]	[]																	
P7																										[]	[]	[]					[]									
P8																																			[]	[]	[]						

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	10	0	42.6	42.6	0
P2	1	3	1.2	11.6	10.6	0.2
P3	2	2	2.4	7	5	0.4
P4	6	5	7.2	20.8	14.8	1.2
P5	15	12	15.4	46.8	31.8	0.4
P6	25	2	25.8	27.8	2.8	0.8
P7	27	4	28	37.8	10.8	1
P8	33	3	33.6	36.6	3.6	0.6
AVG	13.625	5.125	14.2	28.875	15.25	0.575

```
[       OK ] Default/SchedulerTest.FeedBack_1/3 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.FeedBack_2i/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									[]						[]					

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	3	0	4.02	4.02	0
P2	2	6	2.01	17.1	15.1	0.01
P3	4	4	4.03	18.11	14.11	0.03
P4	6	5	7.05	20.12	14.12	1.05
P5	8	2	8.06	14.09	6.09	0.06
AVG	4	4	4.23	14.688	10.688	0.23

```
[       OK ] Default/SchedulerTest.FeedBack_2i/0 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.FeedBack_2i/1
-------------------------------------------------------------------------------
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      |                                 []                  []
-------------------------------------------------------------------------------
```

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	3	0	4.2	4.2	0
P2	2	6	2.1	18	16	0.1
P3	4	4	4.3	19.1	15.1	0.3
P4	6	5	7.5	21.2	15.2	1.5
P5	8	2	8.6	14.9	6.9	0.6
AVG	4	4	4.5	15.48	11.48	0.5

```
[       OK ] Default/SchedulerTest.FeedBack_2i/1 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.FeedBack_2i/2
-------------------------------------------------------------------------------
Process | 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
-------------------------------------------------------------------------------
P1      | []          []  []                          []  []  []  []                                  []  []  []
P2      |     []              []  []
P3      |         []                  []
P4      |                         []      []  []                          []  []
P5      |                                                 []  []  []          []  []  []  []                                                   []  []  []  []  []
P6      |                                                                                                                 []      []
P7      |                                                                                                                     []      []  []  []  []
P8      |                                                                                                                                 []  []  []
-------------------------------------------------------------------------------
```

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	10	0	27.6	27.6	0
P2	1	3	1.05	7.2	6.2	0.05
P3	2	2	2.1	9.3	7.3	0.1
P4	6	5	7.25	20.5	14.5	1.25
P5	15	12	15.45	41.9	26.9	0.45
P6	25	2	27.65	30.75	5.75	2.65
P7	27	4	28.7	33.8	6.8	1.7
P8	33	3	33.85	36.85	3.85	0.85
AVG	13.625	5.125	14.5063	25.9875	12.3625	0.88125

```
[       OK ] Default/SchedulerTest.FeedBack_2i/2 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.FeedBack_2i/3
-------------------------------------------------------------------------------
Process | 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
-------------------------------------------------------------------------------
P1      | []          []  []                          []  []  []  []                                                  []  []  []
P2      |     []              []  []
P3      |         []                  []
P4      |                         []      []  []                          []  []
P5      |                                                 []  []  []          []  []  []  []                                                   []  []  []  []  []
P6      |                                                                                                                 []      []
P7      |                                                                                                                     []      []  []  []
P8      |                                                                                                                                 []  []  []
-------------------------------------------------------------------------------
```

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	10	0	36.2	36.2	0
P2	1	3	1.2	7.8	6.8	0.2
P3	2	2	2.4	10.2	8.2	0.4
P4	6	5	8	22	16	2
P5	15	12	16.8	44.6	29.6	1.8
P6	25	2	26.4	29.8	4.8	1.4
P7	27	4	27.6	33	6	0.6
P8	33	3	36.4	39.4	6.4	3.4
AVG	13.625	5.125	14.85	27.875	14.25	1.225

```
[       OK ] Default/SchedulerTest.FeedBack_2i/3 (0 ms)
```

STRIDE

```
[ RUN      ] Default/SchedulerTest.Stride/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      |             []  []
-------------------------------------------------------------------------------
```

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	4	0	18.14	18.14	0
P2	0	8	1.01	20.15	20.15	1.01
P3	0	6	2.02	14.11	14.11	2.02
P4	0	2	3.03	5.03	5.03	3.03
AVG	0	5	1.515	14.3575	14.3575	1.515

```
[       OK ] Default/SchedulerTest.Stride/0 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.Stride/1
-------------------------------------------------------------------------------
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      |             []  []
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
-------------------------------------------------------------------------------------------------------------
P1   | 0                | 4                | 0                | 19.4             | 19.4             | 0
P2   | 0                | 8                | 1.1              | 21.5             | 21.5             | 1.1
P3   | 0                | 6                | 2.2              | 15.1             | 15.1             | 2.2
P4   | 0                | 2                | 3.3              | 5.3              | 5.3              | 3.3
-------------------------------------------------------------------------------------------------------------
AVG  | 0                | 5                | 1.65             | 15.325           | 15.325           | 1.65
-------------------------------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.Stride/1 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.Stride/2
-------------------------------------------------------------------------------------------------------------------------------------------------------
Process | 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
-------------------------------------------------------------------------------------------------------------------------------------------------------
P1      | []                                                                                                                  []          []      []  []
P2      |     []
P3      |         []                                                                  []                      []          []  []      []
P4      |             []                          []
P5      |                 []                  []          []          []              []          []  []      []      []
P6      |                     []          []          []          []
P7      |                         []  []          []          []
P8      |                             []  []      []      []          []      []  []      []  []  []
-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
-------------------------------------------------------------------------------------
P1   | 0                | 5                | 0                | 40.55            | 40.55            | 0
P2   | 0                | 1                | 1.05             | 2.05             | 2.05             | 1.05
P3   | 0                | 6                | 2.1              | 38.5             | 38.5             | 2.1
P4   | 0                | 2                | 3.15             | 17.75            | 17.75            | 3.15
P5   | 0                | 8                | 4.2              | 34.35            | 34.35            | 4.2
P6   | 0                | 4                | 5.25             | 20.9             | 20.9             | 5.25
P7   | 0                | 3                | 6.3              | 15.65            | 15.65            | 6.3
P8   | 0                | 10               | 7.35             | 28.1             | 28.1             | 7.35
-------------------------------------------------------------------------------------
AVG  | 0                | 4.875            | 3.675            | 24.7313          | 24.7313          | 3.675
-------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.Stride/2 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.Stride/3
-------------------------------------------------------------------------------------------------------------------------------------------------------
Process | 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38
-------------------------------------------------------------------------------------------------------------------------------------------------------
P1      | []                                                                                                                  []          []      []  []
P2      |     []
P3      |         []                                                                  []                      []          []  []      []
P4      |             []                          []
P5      |                 []                  []          []          []              []          []  []      []      []
P6      |                     []          []          []          []
P7      |                         []  []          []          []
P8      |                             []  []      []      []          []      []  []      []  []  []
-------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
-------------------------------------------------------------------------------------
P1   | 0                | 5                | 0                | 45.2             | 45.2             | 0
P2   | 0                | 1                | 1.2              | 2.2              | 2.2              | 1.2
P3   | 0                | 6                | 2.4              | 43               | 43               | 2.4
P4   | 0                | 2                | 3.6              | 20               | 20               | 3.6
P5   | 0                | 8                | 4.8              | 38.4             | 38.4             | 4.8
P6   | 0                | 4                | 6                | 23.6             | 23.6             | 6
P7   | 0                | 3                | 7.2              | 17.6             | 17.6             | 7.2
P8   | 0                | 10               | 8.4              | 31.4             | 31.4             | 8.4
-------------------------------------------------------------------------------------
AVG  | 0                | 4.875            | 4.2              | 27.675           | 27.675           | 4.2
-------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.Stride/3 (0 ms)
```

LOTTERY

```
[ RUN      ] Default/SchedulerTest.Lottery/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      | []              []
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------
Name | Arrival Time     | Service Time     | First Run Time   | Completion Time  | Turn Around Time | Response Time
-------------------------------------------------------------------------------------------------------------
P1   | 0                | 4                | 2.02             | 17.13            | 17.13            | 2.02
P2   | 0                | 8                | 1.01             | 20.14            | 20.14            | 1.01
P3   | 0                | 6                | 3.03             | 15.12            | 15.12            | 3.03
P4   | 0                | 2                | 0                | 5.04             | 5.04             | 0
-------------------------------------------------------------------------------------------------------------
AVG  | 0                | 5                | 1.515            | 14.3575          | 14.3575          | 1.515
-------------------------------------------------------------------------------------------------------------
[       OK ] Default/SchedulerTest.Lotterv/0 (0 ms)
```

```
[ RUN      ] Default/SchedulerTest.Lottery/1
```

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	[]				[]																

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	4	2.2	18.3	18.3	2.2
P2	0	8	1.1	21.4	21.4	1.1
P3	0	6	3.3	16.2	16.2	3.3
P4	0	2	0	5.4	5.4	0
AVG	0	5	1.65	15.325	15.325	1.65

```
[       OK ] Default/SchedulerTest.Lottery/1 (0 ms)
[ RUN      ] Default/SchedulerTest.Lottery/2
```

Process	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
P1												[]													[]								[]						[]	[]
P2			[]																																					
P3																			[]											[]				[]	[]	[]	[]			
P4																	[]			[]																					
P5		[]							[]		[]		[]		[]														[]		[]		[]								
P6					[]					[]																[]	[]														
P7				[]	[]				[]																															
P8	[]					[]	[]						[]	[]							[]	[]	[]	[]				[]													

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	5	11.45	40.35	40.35	11.45
P2	0	1	2.1	3.1	3.1	2.1
P3	0	6	18.75	38.3	38.3	18.75
P4	0	2	16.65	20.8	20.8	16.65
P5	0	8	1.05	34.25	34.25	1.05
P6	0	4	5.2	27.95	27.95	5.2
P7	0	3	3.15	9.3	9.3	3.15
P8	0	10	0	29	29	0
AVG	0	4.875	7.29375	25.3813	25.3813	7.29375

```
[       OK ] Default/SchedulerTest.Lottery/2 (0 ms)
[ RUN      ] Default/SchedulerTest.Lottery/3
```

Process	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
P1												[]													[]								[]						[]	[]
P2			[]																																					
P3																			[]											[]				[]	[]	[]	[]			
P4																	[]			[]																					
P5		[]							[]		[]		[]		[]														[]		[]		[]								
P6					[]					[]																[]	[]														
P7				[]	[]				[]																															
P8	[]					[]	[]						[]	[]							[]	[]	[]	[]				[]													

Name	Arrival Time	Service Time	First Run Time	Completion Time	Turn Around Time	Response Time
P1	0	5	12.8	44.4	44.4	12.8
P2	0	1	2.4	3.4	3.4	2.4
P3	0	6	21	42.2	42.2	21
P4	0	2	18.6	23.2	23.2	18.6
P5	0	8	1.2	38	38	1.2
P6	0	4	5.8	30.8	30.8	5.8
P7	0	3	3.6	10.2	10.2	3.6
P8	0	10	0	32	32	0
AVG	0	4.875	8.175	28.025	28.025	8.175

```
[       OK ] Default/SchedulerTest.Lottery/3 (0 ms)
[----------] 28 tests from Default/SchedulerTest (11 ms total)

[----------] Global test environment tear-down
[==========] 28 tests from 1 test suite ran. (11 ms total)
[  PASSED  ] 28 tests.
```