**운영체제  
Lab 1. CPU Virtualization**

폰트, 로고, 등록 상표, 상징이(가) 표시된 사진

자동 생성된 설명

주 제 : Linux에서 4가지 Scheduler Simulator 구현

교 수 : 최건희

학 과 : 소프트웨어학과

학 번 : 32190789

이 름 : 김승호

마 감 : 25.05.01.

**소스코드 설명**

**1. RR (Round Robin)**

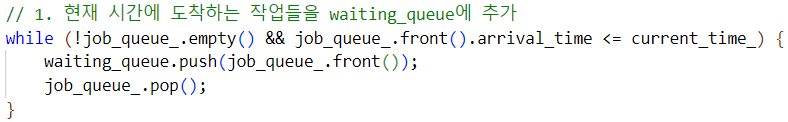
첫 번째 스케줄러는 Round Robin(RR) 기법을 기반으로 작동합니다. 생성자에서 전달된 작업 큐(job\_queue\_)와 context switch 시간(switch\_time\_)을 기반으로, 주어진 time slice만큼 각 작업을 분할하여 순차적으로 실행하는 방식입니다.

* **waiting\_queue**는 현재 시점까지 도착한 작업들을 보관하는 큐입니다.
* **current\_job\_**은 현재 CPU에서 실행 중인 작업이며, left\_slice\_는 해당 작업의 남은 time slice를 관리합니다.
* 매 tick(run() 호출)마다 다음 작업을 1초 실행하며, 이때 남은 시간과 time slice를 감소시키고 상황에 따라 문맥 교환(context switch)을 수행합니다.
* 작업이 완료되면 completion\_time을 기록하고 end\_jobs\_에 저장합니다.

Round Robin 코드는 도착한 작업을 즉시 waiting\_queue에 추가하며, 실행 중인 작업이 없으면 대기열에서 작업을 꺼내 CPU에 할당합니다. Slice가 끝난 작업은 다시 waiting\_queue 뒤로 보냅니다.

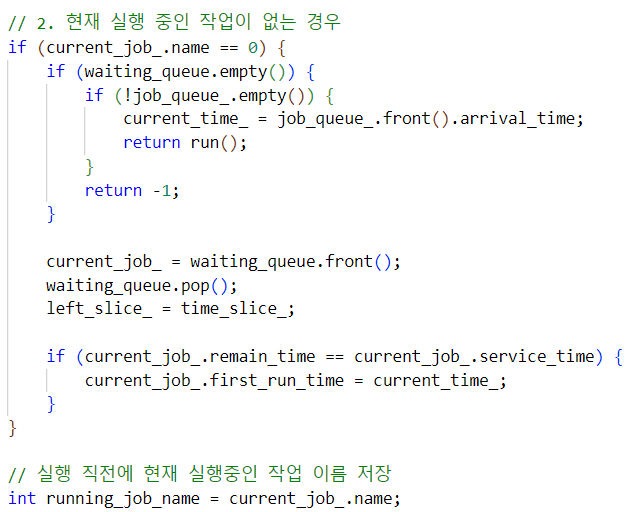
**RR::run() 함수 단계별 설명**

**1. 도착한 작업을 waiting\_queue에 추가**



* 아직 처리 전인 작업들(job\_queue\_) 중에서 현재 시간보다 **도착 시간이 빠르거나 같은 작업들**을 waiting\_queue로 옮긴다.
* **job\_queue\_는 도착 시간 순 정렬**된 큐라고 가정되어 있음.

**2. 현재 실행 중인 작업이 없는 경우**

****

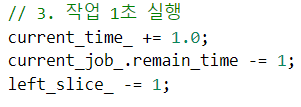
아무 작업도 실행하고 있지 않다면

* waiting\_queue도 비어 있고, job\_queue\_는 아직 작업이 남아 있다면 -> 시간 점프 (current\_time\_을 다음 작업의 도착 시간으로) -> run() 재호출.
* 둘 다 비어 있다면 작업 다 끝난 상태 → -1 반환.

실행할 작업이 있다면

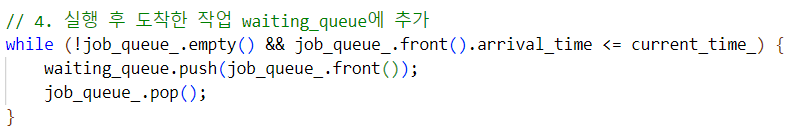
* waiting\_queue에서 하나 꺼내서 현재 작업으로 설정.

**3. 작업을 1초간 실행**



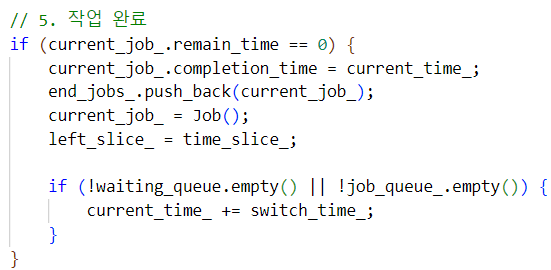
* 1초가 지났으니 현재 시간 증가.
* 현재 작업의 남은 시간, 슬라이스 시간 각각 1씩 감소.

**4. 1초 실행 후 새로 도착한 작업들 추가**



방금 실행하는 동안 도착한 작업도 waiting\_queue에 추가.

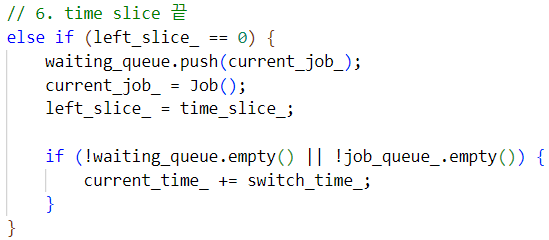
**5. 작업이 끝났다면**



작업 완료되면:

* 완료 시간 기록
* 완료 리스트에 넣음 (end\_jobs\_)
* 현재 작업 초기화
* 문맥 교환 시간 추가 (다음 작업이 있다면)

**6. time slice가 끝났다면**



아직 작업은 끝나지 않았지만 time slice가 다 됐으면:

* 다시 waiting\_queue에 넣고 > 새 작업 준비
* context switch 시간 추가

**7. 실행한 작업 이름 반환**



위 실행 전에 따로 저장한 작업 이름 반환

**Discussion**

**준비**

* 다양한 Workload (A, B)에 대해 다양한 Context Switch Time (0.01, 0.1, 0.2 등)과 여러 스케줄링 알고리즘(RR, FB, Lottery, Stride)을 적용했다.
* RR의 time slice는 1 또는 4로 실험하였다.

**분석**

* **Workload A**, Context Switch Time = **0.01**일 때
  + **Turnaround Time**이 가장 낮은 스케줄러: **RR (time slice = 4)**
  + 이유: RR은 time slice 단위로 context switch가 자주 발생하지만, slice가 4일 경우 context switch 빈도가 낮아져 전환 오버헤드가 줄어든다.
  + 반면, FB는 job starvation이 발생할 수 있고, Lottery나 Stride는 randomness나 정밀도 이슈로 인한 오차가 존재했다.
* **Workload B**의 경우 job들이 중후반에 몰려 있어서 RR보다 FB가 유리하게 작동하기도 했다. 초기 priority가 높은 job이 우선 배치되며 평균 응답 시간이 감소했다.

**배운 점 및 어려웠던 점**

VirtualBox를 처음 사용해봤는데, OS 이미지를 직접 다운로드 받아 직접 가상머신 구동시키는 것이 재미있었습니다. 남는 노트북이 있다면, ubuntu를 VirtualBox를 이용해 구동시켜 개인 서버로 사용해봐도 좋을 것 같다는 생각을 했습니다.

어려웠던 점은, 코드 구현이였습니다. 제공해주신 깃허브를 fork해서 clone한 다음, 로컬에서 작업 후 git push, 가상머신에서 git pull해 구동시켜보는 식으로 진행했는데, 원하는 결과가 잘 나오지 않아서 아쉬웠습니다.