4️⃣ CPU 스케줄링

CPU를 길게 사용하는 CPU bound job과 CPU를 짧게 쓰는 I/O bound job이 있음 → **CPU 스케줄링**

→ interactive한 job에게 적절한 response 제공 요망

CPU scheduler : ready 상태의 프로세스 중에서 이번에 *CPU를 줄 프로세스를 고름*

- 필요한 경우
  1. Running → Blocked (ex. I/O 요청 시스템 콜)
  2. Running → Ready (time interrupt)
  3. Blocked → Ready (I/O 완료 후 interrupt)
  4. terminated

  1, 4는 nonpreemptive (자진반납) / 2, 3은 preemptive (강제로 빼앗음)

Dispatcher : CPU의 *제어권*을 CPU scheduler에 의해 *선택된 프로세스에게 넘김* → context switch

Scheduling criteria (performance Index) : 성능 척도

어떤 스케줄링 알고리즘이 더 좋은지 판단

- CPU utilization

  이용률 : 전체 시간 중 CPU가 *일한 시간의 비율* → 높을수록 좋은 것
- Throughput

  처리량 : 단위시간 당 처리량 → CPU 입장에서 *얼마나 많은 일을 했는가*
- Turnaround Time

  소요시간, 반환시간 : CPU burst에서 I/O burst하러 나간 전체 시간 → *기다린 시간 + CPU 사용시간* 포함
- Waiting Time

  대기시간 : CPU 쓰러와서 *기다린 시간의 합* → queue에서 기다린 시간도 포함
- **R**esponse Time

  응답시간 : 프로세스가 *최초로 CPU를 얻기까지 걸린 시간*

Scheduling Algorithm

1. FCFS (First-Come First-Serve)

- 먼저 온 순서대로 처리 → nonpreemptive
- 효율적이지 못함 - waiting time이 길어질 수 있음

- convoy effect : 앞에 실행시간이 길어서 뒤에 짧은 프로세스가 기다림

2. SJF (Shortest Job First)

- *가장 짧은 CPU burst time*을 가진 프로세스 먼저 스케줄링 → optimal : minimum average waiting time 보장
- nonpreemptive : 일단 CPU 잡으면 CPU burst 완료될 때까지 CPU 선점당하지 않음
- preemptive : 현재 남은 CPU burst time보다 *더 짧은 새로운 프로세스* 도착하면 뺴앗김 → **SRTF**
- *starvation* 발생시킴 → long job은 영원히 CPU를 못 얻을 수 있음
- 누가 짧게 쓰고 누가 길게 쓰는지 처음에 알려주지 않음 → *다음 번 CPU burst time 알 수 없음*

  다음 CPU burst time은 추정만 가능 - 과거의 CPU burst time을 봄

  Exponential averaging

3. Priority Scheduling

- *우선순위 높은 프로세스*에 CPU 먼저 할당 (SJF도 일종의 priority scheduling)
- nonpreemptive, preemptive 둘 다 가능
- starvation 문제 : 우선순위 낮은 건 영원히 실행 안될 수 있음

  → **aging**으로 해결 : 오래 기다리면 기다린 시간만큼 우선순위를 조금씩 높여줌

4. Round-Robin

- 각 프로세스는 *동일한 크기의 할당 시간* (time quantum)을 가진 → preemptive
- 어떤 프로세스도 (n-1)p time unit 이상 기다리지 않음
- q가 크면 FCFS, q가 작으면 context switch overhead가 커짐
- *response time이 짧음*

5. Multilevel queue

- ready queue를 여러 개로 분할 → foreground / background - 독립적인 스케줄링 알고리즘 가짐

  foreground : interactive한 프로세스 (RR) / **background** : batch (FCFS)
- queue에 대한 스케줄링 필요
  - *Fixed priority scheduling* : foreground에 있는 것 모두 실행 후 background처리 → starvation
  - *Time slice* : 각 queue에 CPU time을 적절한 비율로 할당

↑ higest priority

system process

interactive process

interactive editing process

batch process

student process

↓ lowest priority

6. Multilevel feedback queue

- 프로세스가 다른 queue로 이동 가능 → aging을 이 방법으로 구현 가능
- (queue의 수, 각 queue의 scheduling algorithm, process 상위 큐로 보내는 기준, 하위 큐로 보내는 기준, 프로세스가 처음 들어갈 큐를 결정하는 기준) 정의 가능

Multiple-processor scheduling

: CPU 여러 개인 경우

- **homogeneous processor**인 경우

  queue에 한 줄에 세워서 *각 프로세스가 알아서 꺼내가게 함*

  → 반드시 특정 프로세서에서 수행되어야 하는 프로세스가 있는 경우에 문제가 더 복잡해짐

- **load sharing = load balancing**

  일부 프로세서에서 job이 몰리지 않도록 부하를 적절히 공유하는 메커니즘 필요

  → *여러 CPU가 골고루 일하도록* 함

- **symmetric multiprogramming**

  각 프로세서가 각자 *알아서 스케줄링* 결정

- **asymmetric multiprogramming**

  *하나의 프로세서가 시스템 데이터의 접근과 공유를 책임*지고 나머지 프로세서는 거기에 따름

Real-time scheduling

- **hard real-time scheduling**

  hard real-time task는 *정해진 시간 안에 반드시 끝내도록* 스케줄링 해야 함

- **soft hard-time scheduling**

  soft real-time task는 일반 프로세스에 비해 *높은 우선순위*를 갖도록 해야 함

Thread scheduling

- **local scheduling**

  user level thread의 경우 사용자 수준의 thread library에 의해 어떤 thread스케줄 할 지 결정

  → 프로세스 자신이 *내부*에 thread 어떻게 스케줄링 할 지 결정

- **global scheduling**

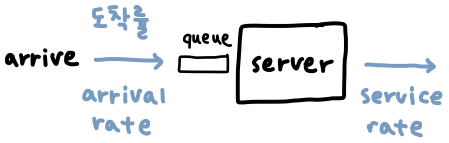
  : kernel level thread의 경우 일반 프로세스와 마찬가지로 커널의 단기 스케줄러가 어떤 thread를 스케줄 할 지 결정

  → 운영체제가 thread의 존재를 알기 때문에 어떤 thread 스케줄 할 지 *직접 결정*

Algorithm Evaluation

- *queueing models*

  확률분포로 주어지는 arrival rate와 service rate 등을 통해 각종 performance index 값을 계산



- *implements(구현) & Measurement(성능 측정)*

  실제 시스템에 알고리즘을 구현하여 실제 작업에 대해서 성능을 측정 비교

- *Simulation (모의 실험)*

  알고리즘을 모의 프로그램으로 작성 후 trace을 입력으로 하여 결과 비교