Chapter 9 - 스케줄링: 비례 배분

비례 배분(Proportional Scheduler) 또는 공정 배분(Fair share)

- 목적: 반환 시간이나 응답 시간을 최적화 하는 것
- 스케줄러가 각 작업에 대해 CPU의 일정 비율을 보장할 수 있게 함

추첨 스케줄링(Lottery Scheduling)

- 추첨권(티켓)을 기반으로 CPU 자원 배분을 결정
- 전체 티켓 중 그 작업이 가진 티켓의 비율이 그 작업이 받아야 할 CPU 몫
- 장점: 구현이 단순하다

```
1  // counter: 당첨자를 발견했는지 추적하는 데 사용됨
2  int counter = 0;
3
4  // winner: 0부터 총 추첨권의 수 사이의 임의의 값을 얻기 위해
5  // 난수 발생기를 호출함
6  int winner = getrandom(0, totaltickets);
7
8  // current: 작업 목록을 탐색하는 데 사용
9  node_t *current = head;
10
11 // 티켓 값 > winner를 만족할 때까지 반복
12 while (current) {
13     counter = counter + current->tickets;
14     if (counter > winner)
15         break; // 당첨자 발견
16     current = current->next;
17 }
18 // "current"는 당첨자를 가리킴: 당첨자가 실행될 수 있도록 준비
```

- 추첨권 화폐를 생성하고 양도 또는 팽창을 통해 그 가치를 실시간으로 조정, 스케줄링을 진행
- 불공정 지표를 사용하여 결과를 확인 가능 → 작업 시간이 충분히 길어야 공정에 가까워짐

- 초기 추첨권을 얼마나 분배해야 하는지 작업의 소요 시간을 모르는 상태에서 결정할 방법 X

보폭 스케줄링(Stride Scheduling)

- 추첨 스케줄링과 유사한 목적을 가지지만 결정론적인 방법으로 뽑히는 스케줄링 방식
- 추첨권 수의 반비례하는 보폭이라는 지표를 가짐

```
current = remove_min(queue);   // pass 값이 최소인 클라이언트로 선택
schedule(current);             // 자원을 타임 퀀텀만큼 선택된 프로세스에게 할당
current->pass += current->stride; // 다음 pass 값을 보폭 값을 이용하여 갱신
insert(queue, current);        // 다시 큐에 저장한다
```

- 추첨 스케줄링과 비교하여 정확도가 높다
- 하지만 프로세스의 상태를 저장해야 한다는 단점 존재
- 또한 새로운 작업이 들어오면 그 작업이 다른 작업만큼 진행될동안 독점하게 된다

결론

- 둘 다 많이 사용되지 않는다
- I/O가 연계되었을 때 동작이 힘들고 추첨권 할당량 문제를 해결하지 못했기 때문
- 그래서 추첨권 할당량을 알 수 있는, 동작 시간이 예측되는 일부 작업에서 유용하게 사용된다