1.

암달의 법칙(Amdahl's Law)은 컴퓨터 시스템에서 일부를 개선할 때 최대 성능을 얼마만큼 향상시킬 수 있는지 계산할 때 사용하는 법칙이다. 더 자세히 살펴보면, 어떤 시스템의 P% 부분을 병렬처리로 전환할 때 그 병렬처리에 사용되는 CPU의 개수나 병렬처리 비율이 변화함에 따라 어느 정도의 최대 성능 향상이 이뤄질 수 있는지 계산하는 법칙이다.

교과서에 나와있는 수식을 풀어서 옮겨적으면 다음과 같다.

$$(\text{성능향상}) \leq \frac{1}{(1-(\text{직렬처리비율}))+\frac{(\text{병렬처리비율})}{(CPU\,\text{코어수})}}$$

(1) 작은 문제 1번은 전체 프로세스의 60%를 병렬로 처리하고 듀얼(2)코어 CPU를 사용한다. 이것을 위의 수식에 대입하면,

$(\text{성능향상}) \leq \frac{1}{(1-0.6)+\frac{0.6}{2}} = \frac{1}{0.7} = 1.\dot{4}2857\dot{1}$, 약 1.4배의 성능 향상을 이룰 수 있다.

(2) 작은 문제 2번은 전체 프로세스의 60%를 병렬로 처리하고 옥타(8)코어 CPU를 사용한다. 이것을 위의 수식에 대입하면,

$(\text{성능향상}) \leq \frac{1}{(1-0.6)+\frac{0.6}{8}} = \frac{1}{0.475} = 2.1052..$, 약 2.1배의 성능 향상을 이룰 수 있다.

2.

(1) 스레드를 경량화된 프로세스를 부를 수 있는 이유는 다음과 같다.

1. 프로세스를 만드는 것은 많은 비용을 요구하지만 스레드는 생성하는데 프로세스 대비 적은 비용을 요구한다. 여기서 비용이라 함은 CPU를 사용하는 시간을 말한다.
2. 프로세스는 다른 프로세스와의 상호작용이 어려운 반면, 스레드는 각각의 stack을 가지고 있다곤 하나 공유된 자원에 접근하는 것은 쉬우므로 스레드간 상호작용이 쉽다. 이로 인해 더 간편하고 저렴한 비용이 소모된다는 점에서 '가볍다' 라고 할 수 있다.
3. 프로세스를 만드는 데에는 새로운 메모리 공간을 할당하고 초기화 해야 하기 때문에 많은 자원을 소모한다. 하지만, 스레드는 훨씬 적은 자원을 소모하기 때문에 프로세스 대비 '가볍다' 라고 할 수 있다.

(2) Thread Pool이란 프로세스가 요청하기 전에 미리 스레드를 만들어두고, 프로세스에서 필요시 할당해주었다가 종료시 회수함으로써 스레드를 관리하는 기법이다. Thread Pool 방식을 통해 스레드를 미리 만들어 놓음으로써 얻을 수 있는 장점은 다음과 같다.

1. 요청시 스레드를 생성하여 제공하는 것보다 미리 만들어진 스레드를 제공하는 것이 훨씬 빠르다.
2. 시스템에서 처리할 수 있는 작업의 양을 Thread Pool을 통해 제한하는 것이 가능하다. 이를 통해 시스템에서 처리 불가능할 정도의 프로세스를 로드하는 일, 즉, 오버로드의 발생 가능성을 낮출 수 있다.
3. 작업 수행 시 다양한 방식으로 수행이 가능하다. 일정 시간의 딜레이 후 실행되거나, 일정 주기를 가지고 반복수행을 하는 등의 예약이 가능하다.

3.

Round Robin(RR) 스케쥴링 알고리즘은 모든 사용자에게 시분할(Time Slicing 또는 Time Quantum) 방법을 이용해 동일한 시간동안 자원을 할당해주는 방법으로 가장 공평한 자원 할당법이라고 할 수 있다.

(1) 위에서 Round Robin 방법을 소개하면서 간략하게 언급했는데, Round Robin의 장점은 자원을 모든 프로세스에게 공평한 시간동안 할당했다가 다시 회수하는 방법이라는 점이 가장 큰 특징이다. 이러한 특징으로 인해 응답성이 짧다는 것이 가장 큰 장점이며, 여러 사용자가 상호작용을 필요로 하는 작업에서는 이러한 응답성이 중요한 요소이기 때문에 많이 사용되는 자원 할당 기법이다.

하지만 이러한 자원 할당 기법은 일정 시간 (자원 할당 시간이 경과되었을 때)마다 다른 프로세스에게 자원을 할당해줘야 하므로 Context Switching이 많이 발생하는 방법이며, 이로 인해 Overhead가 많이 발생하며, 이러한 점은 Round Robin 방법의 단점이다.

(2) Round Robin 방법은 앞서 설명한 대로 일정 시간 간격으로 시분할(Time Quantum 또는 Time Slicing)하여 자원을 할당 및 회수하는 기법이다. 하나의 프로세스에서 다른 프로세스에게 자원을 할당 해 줄 때 Context Switching이 발생하며, 이것은 많은 비용을 요구하는 작업이다.

Time Quantum 값이 작을수록 같은 시간에 이러한 Context Switching이 많이 일어나게 되므로 Overhead가 많이 발생한다는 단점이 있다. 하지만 한 번 자원을 할당받고 다음 할당 받기까지의 주기가 짧기 때문에 응답성 측면에서는 이점을 얻을 수 있다.

이와 반대로 Time Quantum을 좀 더 큰 값으로 설정한다면 한 번 할당됐을 때 하나의 프로세스가 자원을 사용할 수 있는 시간이 늘어나게 되기 때문에 동일 시간 내 Context Switching이 일어나는 횟수는 줄어들게 될 것이다. 하지만 이로 인해 다음 할당까지의 주기가 비례해서 늘어나게 되므로 그만큼 응답성은 나빠질 수밖에 없다.