**[프로세스란?]**

**2. 스케줄링**

1. **프로세스를 스케줄링하기 위한 Queue**

**1-1) Job Queue**

: 현재 시스템 내에 있는 모든 프로세스의 집합

**1-2) Ready Queue**

: 현재 메모리 내에 있으면서 CPU를 잡아서 실행되기를 기다리는 프로세스의 집합

**1-3) Device Queue**

: Device I/O 작업을 대기하고 있는 프로세스의 집합

1. **장기 스케줄러(Long-term Scheduler or Job Scheduler)**

: 한정된 메모리에 많은 프로세스들이 한꺼번에 올라온 경우, 대용량 메모리(일반적으로 디스크)에 임시로 저장된다. 이 pool에 저장되어 있는 프로세스 중 어떤 프로세스에 메모리를 할당하여 Ready Queue로 보낼지 결정하는 역할을 한다.

1. 메모리와 디스크 사이의 스케줄링을 담당
2. 프로세스에 메모리(및 각종 리소스)를 할당(Admit)
3. Degree of Multiprogramming 제어(실행중인 프로세스의 수 제어)
4. 프로세스의 상태 : New -> Ready(in Memory)

Cf) 메모리에는 프로그램이 너무 많이 올라가도, 너무 적게 올라가도 성능이 좋지 않다. 참고로 Time Sharing System 에서는 장기 스케줄러가 없다. 곧바로 메모리에 올라가 Ready 상태가 된다.

1. **단기 스케줄러(Short-term Scheduler or CPU Scheduler)**
2. CPU와 메모리 사이의 스케줄링을 담당
3. Ready Queue에 존재하는 프로세스 중 어떤 프로세스를 Running 시킬지 결정
4. 프로세스에 CPU를 할당(Scheduler Dispatch)
5. 프로세스의 상태 : Ready -> Running -> Waiting -> Ready
6. **중기 스케줄러(Medium-term Scheduler or Swapper)**
7. 여유 공간 마련을 위해 프로세스를 통째로 메모리에서 디스크로 쫓아냄(Swapping)
8. 프로세스에게서 Memory를 Deallocate
9. Degree of Multiprogramming 제어
10. 현 시스템에서 메모리에 너무 많은 프로그램이 동시에 올라가는 것을 조절하는 스케줄러
11. 프로세스의 상태 : Ready -> Suspended

Cf) Suspended(Stopped) : 외부적인 이유로 프로세스의 수행이 정지된 상태로 메모리에서 내려간 상태를 의미한다. 프로세스 전부 디스크로 Swap Out 된다. Blocked 상태는 다른 I/O 작업을 기다리는 상태이기 때문에 스스로 Ready State로 돌아갈 수 있지만, 이 상태는 외부적인 이유로 Suspending 되었기 때문에 스스로 돌아갈 수 없다.

1. **CPU 스케줄러**

: 스케줄링 대상은 Ready Queue에 있는 프로세스들이다.

1. **FCFS(First Come First Served)**
2. **특징**

: 먼저 온 고객을 먼저 서비스해주는 방식

: 비선점형(Non-Preemptive) 스케줄링으로, 일단 CPU를 잡으면 CPU Burst가 완료될 때까지 CPU를 반환하지 않는다. 할당되었던 CPU가 반환될 때만 스케줄링이 이루어진다.

1. **문제점**

: Convoy Effect, 소요시간이 긴 프로세스가 먼저 도달하여 효율성을 낮추는 현상이 발생한다.

1. **SJF(Shortest-Job-First)**
2. **특징**

: 다른 프로세스가 먼저 도착했어도 CPU Burst Time이 짧은 프로세스에게 선 할당한다.

: 비선점형(Non-Preemptive) 스케줄링

1. **문제점**

: Starvation, 효율성을 추구하는 게 가장 중요하지만, 특정 프로세스가 지나치게 차별받으면 안된다. 이 스케줄링은 극단적으로 CPU 사용이 짧은 Job을 선호한다. 그래서 사용 시간이 긴 프로세스는 거의 영원히 CPU를 할당받을 수 없다.

1. **SRTF(Shortest Remaining Time First)**
2. **특징**

: 새로운 프로세스가 도착할 때 마다 새로운 스케줄링이 이루어진다.

: 선점형(Preemptive) 스케줄링으로, 현재 수행중인 프로세스의 남은 Burst Time 보다 더 짧은 CPU Burst Time을 가지는 새로운 프로세스가 도착하면 CPU를 뺏긴다.

1. **문제점**

: Starvation, 새로운 프로세스가 도달할 때 마다 스케줄링을 다시하기 때문에 CPU Burst Time을 측정할 수가 없다.

1. **Priority Scheduling**
2. **특징**

: 우선순위가 가장 높은 프로세스에게 CPU를 할당하는 스케줄링이다. 우선순위란 정수로 표현되고 작은 숫자가 우선순위가 높다.

: 선점형(Preemptive) 스케줄링으로, 더 높은 우선순위의 프로세스가 도착하면 실행중인 프로세스를 멈추고 CPU를 선점한다.

: 비선점형(Non-Preemptive) 스케줄링으로, 더 높은 우선순위의 프로세스가 도착하면 Ready Queue의 Head에 넣는다.

1. **문제점**

: Starvation, 무기한 봉쇄(Indefinite Blocking), 실행 준비는 되어있으나 CPU를 사용하지 못하는 프로세스를 CPU가 무기한 대기하는 상태이다.

1. **해결책**

: Aging, 아무리 우선순위가 낮은 프로세스라도 오래 기다리면 우선순위를 높여줘야 한다.

1. **Round Robin**
2. **특징**

: 현대적인 CPU 스케줄링

: 각 프로세스는 동일한 크기의 할당 시간(Time Quantum)을 갖게 된다.

: 할당 시간이 지나면 프로세스는 선점당하고 Ready Queue의 제일 뒤에 가서 다시 줄을 선다.

: CPU 사용시간이 랜덤한 프로세스들이 섞여있을 경우에 효율적이다.

: 이 방식이 가능한 이유는 프로세스의 Context를 Save 할 수 있기 때문이다.

1. **장점**

: Response Time이 빨라진다. n개의 프로세스가 Ready Queue에 있고 할당시간이 q(Time Quantum)인 경우, 각 프로세스는 q단위로 CPU 시간의 1/n을 얻는다. 즉, 어떤 프로세스도 (n-1)q Time Unit 이상 기다리지 않는다.

: 프로세스가 기다리는 시간이 CPU를 사용할 만큼 증가한다. 즉, 공정한 스케줄링이다.

1. **주의할 점**

: 설정한 Time Quantum이 너무 커지면 FCFS와 같아진다. 또 너무 작아지면 스케줄링 알고리즘의 목적에는 이상적이지만 잦은 Context Switch로 OverHead가 발생할 수 있다. 그렇기 때문에 적당한 Time Quantum을 설정하는 것이 중요하다.