****

**19s Programming Study**

**-기술면접 예상 질문-**

Update 21-07-12

목차

1. [**운영체제2**](#데이터베이스)

**1. 운영체제**

**1.1 프로세스와 스레드**

**● 프로세스(Process)**

- “컴퓨터에서 연속적으로 실행되고 있는 컴퓨터 프로그램”을 뜻하며, 메모리에 올라와 실행되고 있는 프로그램의 인스턴스이다.

- 운영체제로부터 시스템 자원을 할당받는 작업의 단위로, 실행된 프로그램을 의미한다.

**\*\* 참고 \*\***

할당받는 시스템 자원의 예시 : CPU시간, 운영되기 위해 필요한 주소공간, 독립된 메모리 영역

**<특징>**

1) 프로세스는 각각 독립된 메모리 영역(Code,Data,Stack,Heap의 구조)을 할당받는다

2) 기본적으로 프로세스 당 최소 1개의 스레드(메인 스레드)를 갖는다.

3) 각 프로세스는 별도의 주소 공간에서 실행되며, 한 프로세스는 다른 프로세스의 변수나 자료구조에 접근불가

4) 한 프로세스가 다른 프로세스의 자원에 접근하려면 프로세스 간 통신(IPC)를 사용해야한다. 직접 접근 불가.

**● 스레드(Thread)**

- “프로세스 내에서 실행되는 여러 흐름의 단위”를 뜻하며, 프로세스의 특정한 수행경로이다.

- 프로세스가 할당 받은 자원을 이용하는 실행 단위

**<특징>**

1) 스레드는 프로세스 내에서 각각 Stack만 따로 할당받고, Code,Data,Heap 영역은 공유한다.

2) 스레드는 한 프로세스 내에서 동작되는 여러 실행의 흐름으로, 프로세스 내의 주소 공간이나 자원들을 다른 스레드와 공유하면서 실행된다.

3) 한 스레드가 프로세스 자원을 변경하면, 프로세스 내 다른 스레드도 그 변경 결과를 즉시 볼수 있다.

**● 멀티 프로세스(Multi Process)**

- 하나의 응용프로그램을 여러 개의 프로세스로 구성하여 각 프로세스가 하나의 작업을 처리하도록 하는 것.

**<장점>**

1) 여러 개의 자식 프로세스 중 하나에 문제가 발생하면 해당 자식 프로세스만 죽고 다른 영향이 없음.

**<단점>**

1) Context Switching과정에서 캐쉬 메모리 초기화 등 무거운 작업이 진행되고 많은 시간이 소모되는 오버헤드 발생. 각 프로세스는 독립된 메모리를 할당 받아 Context Switching 발생하면 캐쉬 리셋 후 다시 불러와야 한다.

2) 프로세스들은 서로 변수를 공유할 수 없어서 어렵고 복잡한 통신기법(IPC)를 이용해야 한다.

**\*\* 참고 \*\***

**Context Swiching이란?** 동작 중인 프로세스가 대기상태로 변경하면서 해당 프로세스의 상태(Context)를 보관하고, 대기하고 있던 다음 순서의 프로세스가 동작하면서 이전에 보관했던 프로세스의 상태를 복구하는 작업

**● 멀티 스레드(Multi Thread)**

- 하나의 응용프로그램을 여러 개의 스레드로 구성하고 각 스레드로 하여금 하나의 작업을 처리하도록 함.

- 많은 운영체제들이 멀티 프로세싱을 지원하고 있으나, 멀티 스레딩을 기본으로 함. (웹 서버가 대표적 예)

**<장점>**

1) 시스템 자원 소모 감소: 프로세스를 생성해 자원을 할당하는 시스템 콜이 줄어 자원의 효율성 증대

2) 시스템 처리량 증가: 스레드 간 데이터 교환이 간단하고, Context Switching이 빨라 처리비용이 감소

3) 간단한 통신 방법: 프로세스 내 Stack 영역을 제외한 모든 메모리를 공유하므로 프로그램 응답시간이 단축

**<단점>**

1) 디버깅이 까다로우며, 섬세한 설계가 필요하다

2) 다른 프로세스에서 스레드를 제어할 수 없다.

3) 하나의 스레드가 문제가 생기면 전체 프로세스가 영향을 받는다.

4) 멀티 스레드의 경우, 전역변수 등을 함께 사용하므로 자원 공유의 문제가 발생할 수 있음. (동기화 issue)

= 멀티 프로세스 사용 예시: 각 프로세스가 독립적으로 작업을 수행할 수 있는 경우   
= 멀티 스레드 사용 예시: 서로 정보를 공유함으로써 얻는 이득이 큰 경우

**1.2 스케줄링**

**● 프로세스를 스케줄링하기 위한 queue**

1) Job Queue: 현재 시스템 내에 있는 모든 프로세스의 집합

2) Ready Queue: 현재 메모리 내에 있으면서 CPU를 잡아서 실행되기를 기다리는 프로세스의 집합

3) Device Queue: Device I/O작업을 대기하고 있는 프로세스의 집합

**● 스케줄러의 종류**

1) 장기 스케줄러

- 대용량 메모리에 임시로 저장된 프로세스 중 어떤 것에 메모리를 할당하여 Ready Queue로 보낼지 결정.

- 메모리와 디스크 사이의 스케줄링을 담당하며, 프로세스에 메모리를 할당함.

- 프로세스의 상태 : New -> Ready

2) 단기 스케줄러

- CPU와 메모리 사이의 스케줄링 담당

- Ready Queue에 존재하는 프로세스 중 어떤 프로세스를 running 시킬지 결정

- 프로세스 상태: Ready –> Running ->Waiting -> Ready

3) 중기 스케줄러

- 여유공간 마련을 위해 프로세스를 통째로 메모리에서 디스크로 보냄.

- 현 시스템에서 메모리에 너무 많은 프로그램이 올라가는 것을 조절하는 스케줄러

- 프로세스 상태: Ready -> Suspended

**● CPU 스케줄러**

1) 선점형 스케줄러: 한 프로세스가 CPU를 점유 중일 때, 우선 순위가 높은 프로세스가 CPU를 차지할 수 있으나, 선점이 일어날 경우 오버헤드가 발생하며 처리시간 예측이 힘들다.

**- SRT(Shortest Remaining Time) 스케줄링:** 짧은 시간 순서대로 프로세스를 수행한다. 남은 처리 시간이 더 짧은 프로세스가 Ready 큐에 들어오면 그 프로세스가 바로 선점됨. 아래에 소개할 SJF의 선점 버전이라고 할 수 있다.

**- 라운드로빈(Round-Robin)스케줄링:** 각 프로세스는 같은 크기의 CPU 시간을 할당 받고 선입선출에 의해 행된다. 할당시간이 너무 크면 선입선출과 다를 바가 없어지고, 너무 작으면 오버헤드가 너무 커진다.

**- 다단계 큐(Multi-level Queue) 스케줄링:** Ready큐를 여러 개 사용하는 기법. 각각의 큐는 자신의 스케줄링 알고리즘을 수행하며, 큐와 큐 사이에도 우선순위를 부여한다.

**- 다단계 피드백 큐 스케줄링:** 다단계 큐와 비슷하나 프로세스들이 큐를 이동할 수 있다.

2) 비선점형 스케줄러: 프로세스가 작업이 완료 된 후, 스스로 CPU를 놓아주는 시점에만 스케줄링이 일어난다.

**-**  **HRN(Highest response ratio next) 스케줄링:** 긴 작업과 짧은 작업간의 지나친 불평등을 어느 정도 보완한 기법. 수행시간의 길이와 대기 시간을 모두 고려해 우선순위를 정한다.

**- SJF(Shortest Job First) 스케줄링:** 큐 안에 있는 프로세스 중 수행시간이 짧은 것을 먼저 수행. 평균 대기 시간을 감소시킨다.

**- 우선순위(priority) 스케줄링:** 프로세스에게 우선순위를 정적, 혹은 동적으로 부여하여 우선순위가 높은 순서대로 처리한다. 동적으로 부여할 경우, 구현이 복잡하고 오버헤드가 많다는 단점이 있으나, 시스템의 응답속도를 증가시킨다.

**- 기한부(Deadline) 스케줄링:** 작업을 명시된 시간이나 기한 내에 완료하도록 계획.

**- FIFO 스케줄링:** 프로세스들은 Ready큐에 도착한 순서대로 CPU를 할당 받는다. 작업 완료 시간을 예측하기 매우 용이하다. 하지만 덜 중요한 작업이 중요한 작업을 기다리게 할 수도 있다.