2023년 2월 27일
Thread, Thread Pool
보고서

Thread, Thread Pool

임원호

-목차-

[참고 문헌]

Ⅰ. Thread

1. Thread란

Thead는 프로그램의 여러 부분을 동시에 실행할 수 있는 가벼운 프로세스로, 프로그램이 동시에 여러 작업을 수행 할 수 있도록 하여, 성능과 응답성을 향상시킨다. 프로그래밍 언어와 사용 중인 운영 체제에 따라 여러 가지 방법으로 생성될 수 있다. Thread를 만드는 일반적인 방법은 new를 통한 Thread클래스를 객체로 생성하는 방법이며, 이 객체는 공유 데이터에 대한 액세스 동기화, 실행제어와 같은 작업을 위한 고급 구성을 가지게 된다. 일반적으로 하나의 프로그램은 하나의 Thread를 가지고 있다. Thread는 웹 서버, 멀티미디어 애플리케이션, 과학 시뮬레이션 및 기타 많은 애플리케이션에서 사용될 수 있다. 본 보고서에서는 웹 서버를 중심으로 작성한다.

2. Process와 Thread의 비교

Thead는 동시에 실행 할 수 있다는 점에서, 프로세스와 유사하지만, 프로세스와 달리 동일한 메모리 공간을 공유하기 때문에 프로세스 간 통신보다 Thread 간 통신 및 데이터

공유가 훨씬 쉽고 빠르게 이루어진다. 하지만, 멀티 프로세스에서 각 프로세스는 독립적으로 실행 되며, 각각 별개의 메모리를 차지하고 있어, 메모리를 공유해서 사용할 수 없다. 다음은 프로세스와 Thread의 차이점을 나타낸다.

1. 메모리 공간 : Thread는 동일한 메모리 공간을 공유하지만, 프로세스는 자체적으로 별도의 메모리 공간을 가지고 있다. 이는 Thread간 통신과 데이터 공유가 프로세스 간 통신보다 쉽고 빠르다는 것을 의미한다.

2. 생성 : 새로운 프로세스를 만드는 것은 새로운 Thread를 만드는 것보다 비용이 많이 들게 된다. 새 프로세스를 만드는 것은 모든 리소스와 메모리를 포함한 상위 프로세스를 복제하는 것을 포함하는 반면, 새 Thread를 만드는 것은 스택만 복제하면 되기 때문이다.

3. 제어 : 프로세스는 독립적인 실행 단위이며 서로 독립적으로 제어할 수 있는 반면 Thread는 상위 프로세스에 종속되어 동일한 프로세스 제어 블록을 공유한다.

4. 컨텍스트 스위칭[1] : 프로세스 간의 컨텍스트 스위칭이 Thread간의 컨텍스트 스위칭 보다 느리다. 프로세스 간 컨텍스트 스위칭에는 저네 프로세스 컨텍스트를 저장 및 복원하는 작업이 수반되는 반면, Thread 간 컨텍스트 스위칭에는 Thread 컨텍스트만 저장 및 복원하는 작업이 수반되기 때문이다. (오버헤드[2] 주의)

5. 자원 공유 : 프로세스가 서로 분리되어 있으므로 자원을 공유하려면 프로세스 간 통신(IPC) 메커니즘을 사용해야 하는 반면, Thread는 자원을 더 쉽고 효율적으로 공유할 수 있다.

3. Socket Programming 에서의 Thread

소켓 프로그래밍에서, Thread는 여러 클라이언트와 서버 간의 동시 네트워크 통신을 가능하게 하는데, 사용될 수 있다. 서버는 일반적으로 네트워크 소켓에서, 들어오는 여러개의 클라이언트 연결을 수신하며, 클라이언트가 연결될 때 서버는 해당 클라이언트와의 통신을 처리하기 위해 다중 Thread를 사용한다. 즉 1 : 1의 상황에서도, 1 : N의 상황에서도 서버는 Thread를 사용한다. 만약 서버가 여러대의 클라이언트의 요청을 받을 때에는, 서버는 Thread라는 서버 대리인을 생성하고, 클라이언트와 연결시켜 주면 된다. 허나, Thread를 너무 과도하게 생성하거나, Thread의 연결이 끊어짐없이 Stateful하게 된다면, 서버는 자원을 낭비하게 되어, 교착상태에 들어설 수 있다. 프로세스와 달리 Thread는 서버의 자원을 쉽게 접근하며 공유한다. 만약 A Thread가 하나의 자원을 계속해서

---

[1] 컨텍스트 스위칭이란, 여러 Thread 또는 프로세스가 하나의 프로세서를 공유할 수 있도록, 현재 실행 중인 Thread 또는 프로세스의 상태를 저장하고, 이전에 중단된 Thread 또는 프로세스의 상태를 복원하는 프로세스이다. 즉 Thread 또는 프로세스의 레지스터 값과 다른 실행 컨텍스트 정보를 메모리에 저장한 다음, 저장된 레지스터 값과 실행할 다음 Thread 또는 프로세스의 실행 컨텍스트 정보를 복원한다. 작업이 빈번하게 일어날 경우 오버헤드가 빠르게 증가 할 수 있다. 그래서 각 운영 체제와 애플리케이션은 효율적인 스케줄링 알고리즘을 사용하고, 자원 사용을 최적화하여 컨텍스트 스위칭을 최대한 최소화한다.

[2] 오버 헤드는 컴퓨터에서 작업을 수행하는 데 필요한 추가 비용 또는 자원을 말한다. 자바와 같은 객체 지향 언어에서는 가비지 컬렉션과 가방 메서드 디스패쳐와 같은 기능으로 다른 언어들에 비해 추가적인 오버헤드를 가지게 된다.

쥐고 있는 상태라면, B Thread는 A Thread가 자원 할당을 해제하기 전까지, 계속해서 기다리는 교착상태가 된다. 이런 교착상태를 방지하기 위해 뮤텍스, 세마포어가 존재한다.

4. 뮤텍스, 세마포어와 교착상태

뮤텍스와 세마포어는 서버의 공유 자원에 대한 액세스를 제어하고, 레이스 컨디션을 방지한다. 동시 프로그래밍에서 사용되는 동기화 메커니즘이다.
뮤텍스는 상호 배제의 줄임말로, 공유 자원에 대해 상호 배제를 시행하는 데 사용된다. 한 번에 하나의 Thread만 자원에 액세스할 수 있음을 의미한다. Thread가 자원을 사용하는데, 뮤텍스에게 허락을 받게 디면, 그 Thread만 자원의 독점적인 액세스 권한을 얻게되며, Thread가 다시 자원을 반납하기 전까지, 다른 Thread는 자원 할당을 기다려야 한다.
세마포어는 제한된 용량으로 공유 리소스에 대한 액세스를 제어하는 데 사용되는 신호 메커니즘이다. 세마포어는 한 번에 리소스에 액세스할 수 있는 Thread 수를 제한하거나, Thread가 자원에 액세스 할 수 있는 순서를 조정하는데 사용된다. Thread가 세마포어를 가질 때 Count값이 증가하게 되고, 다시 반납할 때 Count값이 감소하는 등의 기능을 가지고 있다.
일반적으로 뮤텍스와 세마포어는 동시 프로그래밍에서 공유 자원에 대한 액세스를 제어하는 데 사용되는 도구들이며, 이들 사이의 선택은 응용 프로그램에서 특정 요구 사항에 따라 다르게 설계될 수 있다. 뮤텍스는 일반적으로 코드의 중요한 부분을 보호하는 데, 더 간단하고 효율적인 반면, 세마포어는 더 유연하고 더 복잡한 동기화 패턴을 구현하는 데 사용될 수 있다. 밑은 뮤텍스와 세마포어를 사용하지 않고, 동시 프로그래밍을 할 때 생기는 문제들이다.

1. 상호 배제 : 프로그램에서 하나 이상의 자원이 공유 불가능 모드로 유지되어야 한다. 즉 한번에 하나의 프로세스 또는 Thread만 자원을 사용할 수 있다. 자원 할당을 요청할 경우 그것을 가진 프로세스 또는 Thread가 자원 할당을 해제하기 전까지, 기다려야한다.

2. 점유 대기 : 프로세스 또는 Thread가 하나 이상의 리소스를 보유하고 추가 자원을 획득하기 위해서는 대기하고 있어야한다. 자원 할당을 해제하기 전까지 대기하고 있어야한다. 여기서 Thread에게 우선순위를 부여할 경우 우선순위에 따라 자원을 할당 받을 수 있지만, 낮은 우선순위의 Thread가 자원을 할당받게 된다면, 우선순위 반전 교착상태가 발생할 수 있으므로 주의해야한다.

3. 비선점 : 자원을 선점하지 못한다. 즉 자원을 보유하고 있는 프로세스나 Thread가 문제를 해결하기 전에 강제로 자원 할당을 해제할 수 없다.

4. 순환 대기 : 둘 이상의 프로세스 또는 Thread로 구성된 원형 체인이 서로가 보유한 자원을 대기하고 있다. 각 프로세스 또는 Thread가 대기 중인 리소스를 보유하고 있는 경우 이 상태로 인해 교착 상태가 발생할 수 있다.

---

1. 자원 경합 상태 : 둘 이상의 Thread가 공유 자원에 대한 배타적 액세스를 기다리고 있지만, 현재 자원 할당을 해제하려는 Thread가 없을 때 생긴다.

2. 순서 교착 상태 : 둘 이사의 Thread가 보유한 자원을 해제하기 위해 서로 대기하고 있으며, 이로 인해 해당 Thread가 진행되지 않도록 하는 순환 종속성이 생성된다. 이 문제는 순서대로 자원을 획득하지만, 다른 순서로 자원을 반납할 때 생긴다.

3. 라이브락 : 교착 상태와 유사하게 라이브락은 두 개 이상의 Thread가 보유한 자원을 반환하기 위해 서로 대기할 때 발생하지만, 차단되는 대신 불필요한 작업이나 Busy-Wait를 계속 수행해 어느 Thread도 진행하지 못하게 한다.

4. 우선 순위 반전 교착 상태 : 낮은 우선순위를 가진 Thread가 높은 우선순위를 가진 Thread에 필요한 자원을 보유하고 있지만, 낮은 우선순위를 가진 Thread가 뮤텍스 또는 세마포어를 보유하고 있기 때문에 선점되지 않을 때 발생한다. 따라서, 우선 순위가 낮은 Thread가 자원을 해제할 때까지 높은 우선 순위 Thread가 차단된다.

Ⅱ. Thread Pool

1. Thread Pool이란

Thread Pool은 일정한 수의 작업자 Thread를 미리 생성한 다음 일련의 작업을 수행하는데, 사용되는 동시 프로그래밍에 사용되는 설계 패턴이다. Thread Pool은 각 작업에 대한 Thread를 생성하고, 파괴하는 오버헤드를 줄임으로써 동시 프로그램의 성능과 확장성을 향상시키는 데 도움을 준다. 여러 Thread에 걸쳐 워크로드의 균형을 맞추고, 동시 프로그램의 전반적인 성능과 확장성을 향상시키는 데 도움이 될 수 있다. 아래는 Thread Pool의 주요 기능과 장점들이다.

1. 고정된 작업자 Thread 수 : Thread Pool에는 프로그램의 라이프 사이클 동안 미리 생성되고, 유지되는 고정된 수의 작업자 Thread가 있다. 이렇게 하면 각 작업에 대한 Thread를 만들고, 삭제하는데 드는 오버헤드를 줄인다.

2. 작업 대기열 : Thread Pool에는 작업자 Thread에서 실행할 작업 집합을 보관하는 작업 대기열이 존재한다. 작업은 Thread Pool에 요청이 오면, 대기열에 추가되고 작업자 Thread는 사용가능해지면 대기열에서 작업을 가져온다,

3. 로드 밸런싱 : Thread Pool은 사용가능한 Thread에 작업을 균등하게 분산하여 여러 작업자 Thread간에 워크로드 균형을 조정하는 데 도움이 된다.

4. 성능 향상 : Thread Pool은 각 작업에 대한 Thread를 만들고, 파괴하는 오버헤드를 줄여 프로그램의 성능을 향상시킨다. 실행 시간이 단축되고, 전체적인 확장성이 향상된다.

5. 자원 관리 : Thread Pool은 프로그램에 의해 생성되고, 사용되는 Thread 수를 제한함으로 CPU 및 메모리와 같은 시스템 자원을 관리하는데 도움을 준다.

6. 사용자 지정 설정 : Thread Pool은 작업자 Thread 수, 작업 대기열 크기, Thread 생성 및 제거 규칙을 수정 및 설정이 가능하며, 자원 활용을 최적화 할 수 있다.

2. ExecutorService

자바에서는 Thread Pool과 같은 기능을 ExecutorService클래스를 통해 제공한다. 이를 자세하게 알아보면, ExecutorService클래스를 통해 Java에서 Thread Pool을 보다 유연하고, 강력하게 사용할 수 있는 고급 API이다. 다음은 ExecutorService 클래스에서 제공하는 기능과 장점들이다. ExecutorService 클래스를 사용하려면, 일반적으로 newFixed와 같이 Executors 클래스에서 제공하는 정적 Factory Method 중 하나를 사용해 클래스의 인스턴스를 생성한다.

1. Thread Pool Management : ExecutorService 클래스는 Thread Pool에서 작업자 Thread에서 작업자 Thread를 생성, 종료 및 재사용을 관리하고 제어할 수 있는 방법을 제공한다.

2. Task Execution : ExecutorService 클래스는 실행 가능 또는 호출 가능 개체로 작업을 실행 위해 Thread Pool에 제공하는 방법을 제공한다.

3. Task Queuing : ExecutorService 클래스는 Thread Pool에 모든 작업자 Thread가 사용 중인 경우 실행을 위해 작업을 대기열에 넣을 수 있는 방법을 제공한다. 이를 통해 시스템에 너무 많은 Thread를 로드하지 않고, 작업을 적시에 효율적으로 실행할 수 있다.

4. Scheduling and Ordering : ExecutorService 클래스는 특정 시간 또는 특정 지연 시간에 실행할 일을 예약하고, 일이 실행되는 순서를 제어할 수 있는 방법을 제공한다.

5. Future Results : ExecutorService 클래스는 미래 개체 또는 콜백 메소드를 통해 제출된 작업의 결과를 검색하는 방법을 제공한다.

6. Customizable Settings : ExecutorService 클래스는 작업자 Thread 수, 최대 대기열 크기, Thread 생성 및 제거 정책과 같은 Thread Pool의 설정을 사용자가 커스텀 할 수 있는 방법을 제공한다.

[참고 문헌**]**

Java API Document