디지털컨버전스 기반 UXUI Front 전문 개발자 양성과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 - JungHyun LEE(이정현)
akdl911215@naver.com

1) Thread란??

링크 :

하나의 프로세스 내부에서 독립적으로 실행되는 하나의 작업 단위를 말하며, 세부적으로 운영체제에 의해 관리되는 하나의 작업 혹은 태스크를 의미합니다. 스레드와 태스크(혹은 작업)ㅇㅡㄴ 바꾸어 사용해도 무관합니다.

1.JVM에 의해 하나의 프로세스가 발생하고 main() 안의 실행문들이 하나의 스레드입니다.
2.main() 이외의 또 다른 스레드를 만들려면 Thread 클래스를 상속하거나 Runnable 인터페이스를 구현합니다.
3.다중 스레드 작ㅐㅓ 시에는 각 스레드 끼리 정보를 주고받을 수 있어 처리 과정의 오류를 줄일 수 있습니다.
4. 프로세스끼리는 정보를 주고받을 수 없습니다.

결국은, Thread는 시스템 작업을 효율적으로 관리하기 위해서 사용한다. 멀트프로세스(다양한 메인들) 사용하지 않고 한개의 프로세스(한개의 메인)를 실행하게 되면 자원을 할당하는 과정도 줄어들 뿐더러 프로세스 컨텍스트 스위칭 하는 것보다 오버헤드를 더 줄일 수 있게 된다.

뿐만 아니라 프로세스간의 통신 비용보다 하나의 프로세스 내에서 여러 스레드간의 여러 스레드간의 통신 비용이 훨씬 적으므로 작업들 간의 통신 부담을 줄일 수 있게 된다.

그리고 멀티 스레딩을 너무 자주 사용하게 되면 컨텍스트 스위칭의 비용이 상당히 높기 때문에 오히려 시스템 성능 저하를 초래 할 수도 있고 메모리가 공유되기 때문에 안정성 및 보안 (간단하게 유지보수)에 취약해 질 수 있을것같다.

2) 컨텍스트 스위치(Context Switching)란??

링크 :

멀티프로세스 환경에서 CPU가 어떤 하나의 프로세스를 실행하고 있는상태에서 인터럽트 요청에 의해 다음 우선순위의 프로세스가 실행되어야 할 때 기존의 프로세스의 상태 또는 레지스터 값(Context)을 저장하고 CPU가 다음 프로세스를 수행하도록 새로운 프로세스의 상태 또는 레지스터 값(Context)를 교체하는 작업을 Context Switch(Context Switching)라고 한다.

정리하자면, CPU는 기본적으로 한번에 한가지의 일만 할 수 있기때문이다. 그렇지만 우리가 여러가지를 할 수 있다고 느끼는 이유는 한가지일을 짧은시간에 처리하고 번걸아가면서 하기 때문에 멀티테스킹이 되는 듯한 느낌을 받는 것이다. 그리고 다양한 일들을 처리하기 위해 교체하는 작업을 컨텍스트 스위치라고 부르는듯 하다.

3) 오버헤드(Overhead)란??

링크 :

오버헤드란 어떤 처리를 하기 위해 들어가는 간접적인 처리 시간/메모리 등을 말한다.

'간접적인'이라는 표현이 들어가는 이유는 목적 달성에 본질적인 것은 아니지만 필요한 자원이기 때문이다.

ex) A라는 작업을 단순히 처리하고 실행하는 시간은 5초 걸리는데, 안정성을 고려하고 부가적인 B라는 처리를 추가한 결과 처리시간이 10초 걸렸다면, 오버헤드는 5초가 된다. 또한, 이 처리에서 B를 개선해서 처리한 결과, 처리시간이 8초가 된다면 , 이 경우 오버헤드가 2초 단축되었다고 한다.

4) 데드락(Dead-lock)란??

링크 :

데드락(Dead-lock)은 말 그대로 프로세스가 죽어버린것이다. 두 개의 스레드가 있는데 서로 정보를 주기위해 영원히 기다리는 상황을 이야기합니다.

데드락을 검색하다보니깐 흥미로운 비유의 글이 있었다.

"데드락은 예전부터 '식사하는 철학자'문제로 널리 알려져 왔다. 다섯 명의 철학자가 중국 음식점에 저녁 식사를 하러 가서 동그런 테이블에 앉았다. 테이블에는 다섯 개의 젓가락(다섯 쌍이 아닌 다섯 개)이 개인별 접시 사이에 하나ㅅㅠㄱ 놓여있다. 철학자는 '먹는' 동작과 '생각하는' 동작을 차례대로 반복한다. 먹는 동안에는 접시 양쪽에 있는 젓가락 두 개를 모아 한ㅆ・ㅇ을 만들어야 자신의 접시에 놓인 음식을 먹을 수 있고, 음식을 먹은 이후에는 젓가락을 다시 양쪽에 하나씩 내려 놓고 생각을 시작한다.
(중략)
모든 철학자가 각기 자기 왼쪽에 있는 젓가락을 집은 다음 오른쪽 젓가락을 사용할 수 있을 떄까지 기다렸다가 오른쪽 젓가락을 집어서 식사를 한다면, 모든 철학자가 더 이상 먹지 못하는 상황에 다다를 수 있다. 철학자 모두가 먹지 못하는 상황은 음식을 먹는데 필요한 자원을 모두 다른 곳에서 확보하고 놓지 않기 때문에 모두가 서로 상대방이 자원을 놓기만을 기다리는, 이른바 '데드락'이 걸린다."

5) Worker impllements Runnable

링크 :
https://github.com/akdl911215/GroupStudy/blob/main/junghyunlee/java_work/src/Eighteenth/FirstThreadTest.java

```
public class FirstThreadTest {
    public static void main(String[] args) {
```

```
Thread t1 = new Thread(new Worker( name: "대머리독수리"));
System.out.println("준비");
Thread t2 = new Thread(new Worker( name: "대갈장군"));
Thread t3 = new Thread(new Worker( name: "뿌뿌뿡"));
```

실행이 되면 메인에서 부터 실행되며 new Thread를 읽고나면 Worker umplements Runnable을 실행하게 된다.

```
class Worker implements Runnable {
```

```
private final static Random generator = new Random();

public Worker(String name) {
    this.name = name;
    sleepTime = generator.nextInt( bound: 3000) + 500;
}
```

sleepTime 에는 0.5~3.5까지의 랜덤값이 할당된다.

실헹하고나면 implements 의 Runnable의 run() 을 실행한다
run을 실행하면 랜덤값만큼 sleep하고 println이 출력된다.

sleep을 사용한 이유는 thread의 실행순서를 보여주기 위한것이라는 생각이 든다.

```
@Override
public void run() {
    System.out.println("시작");
    try {
        Thread.sleep(sleepTime);
    } catch (InterruptedException e) {

    }

    System.out.println(name + "SCV가 미네랄을 채취했습니다.");
}
}
```

6) SecondThreadTest

링크 :
https://github.com/akdl911215/GroupStudy/blob/main/junghyunlee/java_work/src/Eighteenth/SecondThreadTest.java

```
public static void main(String[] args) {
```

메인이 실행되면 Thread는 읽히고 println이 출력된다.

```
System.out.println("main() 프로세스 실행중");
```

그리고나서 SecondThreadTest가 실행된다.

```
public class SecondThreadTest implements Runnable {
    private String name;

    public SecondThreadTest(String name) { this.name = name; }

    @Override
    public void run() {
        Random random = new Random();

        for(int i = 1; i < 3; i++) {
            System.out.println(name + ": " + random.nextInt( bound: 100));

            try {
                Thread.sleep( millis: 500);
            } catch(InterruptedException e) {

            }
        }
    }
```

run에서 포문으로 2회 반복된다. 그리고 Thread두개가 경쟁을 해서 출력된다. 출력값에는 0~99까지 랜덤으로 출력된다.

main() 프로세스 실행중이 실행되고 0.05초후에 출력된다.

7) ThirdThreadTest

링크 :
https://github.com/akdl911215/GroupStudy/blob/main/junghyunlee/java_work/src/Eighteenth/TirdThreadTest.java

```
public static void main(String[] args) {
```

메인이 실행되면 Thread는 읽히면서 0.8초 sleep 전까지 출력된다. t1.start만 출력된다. t2.start는 Thread라 실행은 되지만 sleep 이후이기 때문에 출력이 안된다.

```
try {
    Thread.sleep( millis: 800);
} catch (InterruptedException e) {

}

t2.start();
System.out.println("main() 실행중");
```

슬립으로 인해 시간이 지나서 main()이 실행되고 t1과 t2가 남은수만큼 출력된다.

```
// 데몬 프로세스: init(1) 프로세스가 직접 개입하여
//              심판을 내리기 전까지는 죽지 않는 불멸의 프로세스
//              (그래서 서비스 운영에서 밥먹듯이 사용됨)
t2.setDaemon(true);
```

데몬프로세스는 결과값과 상관없는 것으로 보아서 설명을 위해 존재하는 듯하다.

8) 질문 Q&A

- FirstThreadTest 의 Worker안에 private final static Random generator = new Random(); 이 있습니다.

여기서 궁금한 점은 랜덤을 굳이 static으로 선언한 이유입니다.

static을 사용하면 static 키워드를 통해 생성된 정적멤버들은 heap영역이 아닌 static 영역에 할당됩니다. static 영역에 할당된 메모리는 모든 객체가 공유하여 하나의 멤버를 어디서든지 참조할 수 있는 장점을 가지게 됩니다. 그렇기 때문에 static으로 선언을 한 이유는 밑에서 나오는 랜덤값을 빠르게 전달하기 위해서 라는 생각이 됩니다. 하지만 static을 사용하면서 빠르게 전달해야 하는 이유가 있는건가요???

링크 : https://github.com/akdl911215/GroupStudy/blob/main/junghyunlee/java_work/src/Eighteenth/FirstThreadTest.java

- 태스크(Tesk)에 대해서 다시 설명부탁드립니다.