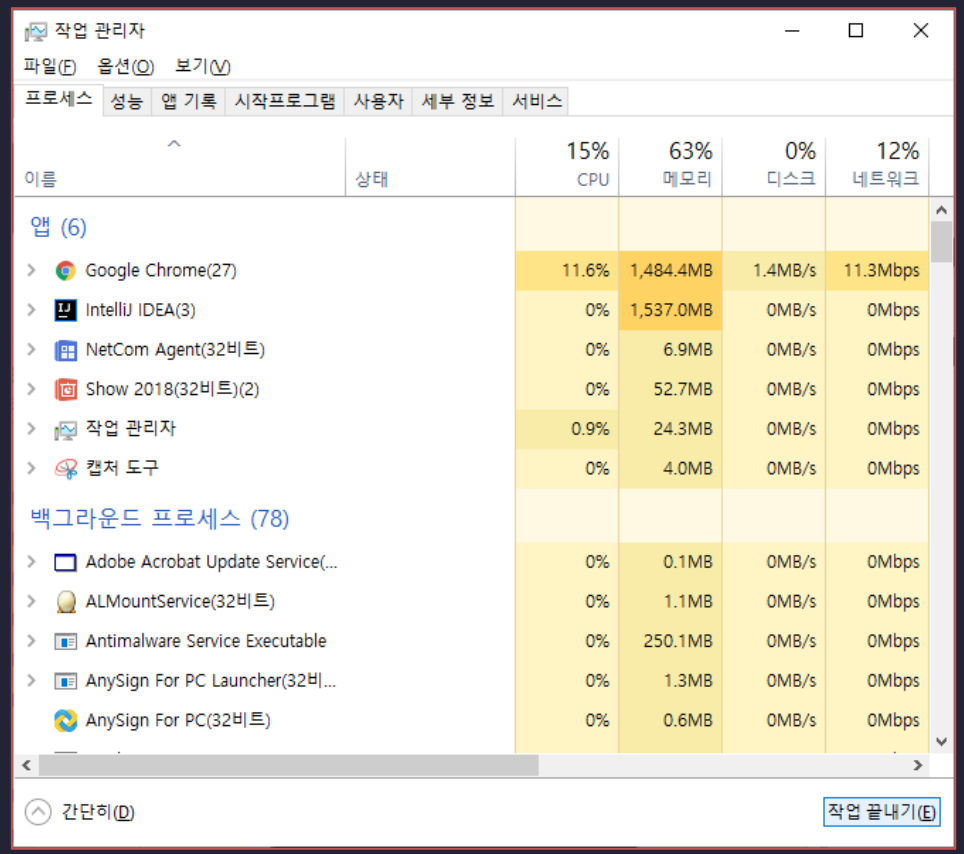
(디지털 컨버전스) 스마트 콘텐츠와 웹 융합 응용 SW개발자 양성과정

훈련기간 : 2021.05.07 ~ 2021.12.08

KH정보교육원 | 전승리

프로세스와 스레드



프로세스 :
운영체제에서는 실행중인 하나의 어플리케이션을 프로세스라고 한다.
운영체제로부터 실행에 필요한 메모리를 할당받아 어플리케이션의 코드를 실행.
프로세스는 CPU의 작업을 추상화해놓은것 생각하면 편하게 접근할수있다.

쓰레드 :
프로세스를 이루는 코드의 실행흐름이며 하나의 스레드는 하나의 코드 실행흐름이다.

멀티 쓰레드 :
2개이상의 쓰레드가 한 프로세스 내에서 처리되는 것을 말한다.
하나의 프로세스 내부에 존재하기 때문에 하나의 스레드가 예외를 던지면 프로세스 하나가 아예 동작하지 않을 수 있다. (예외 처리 중요)

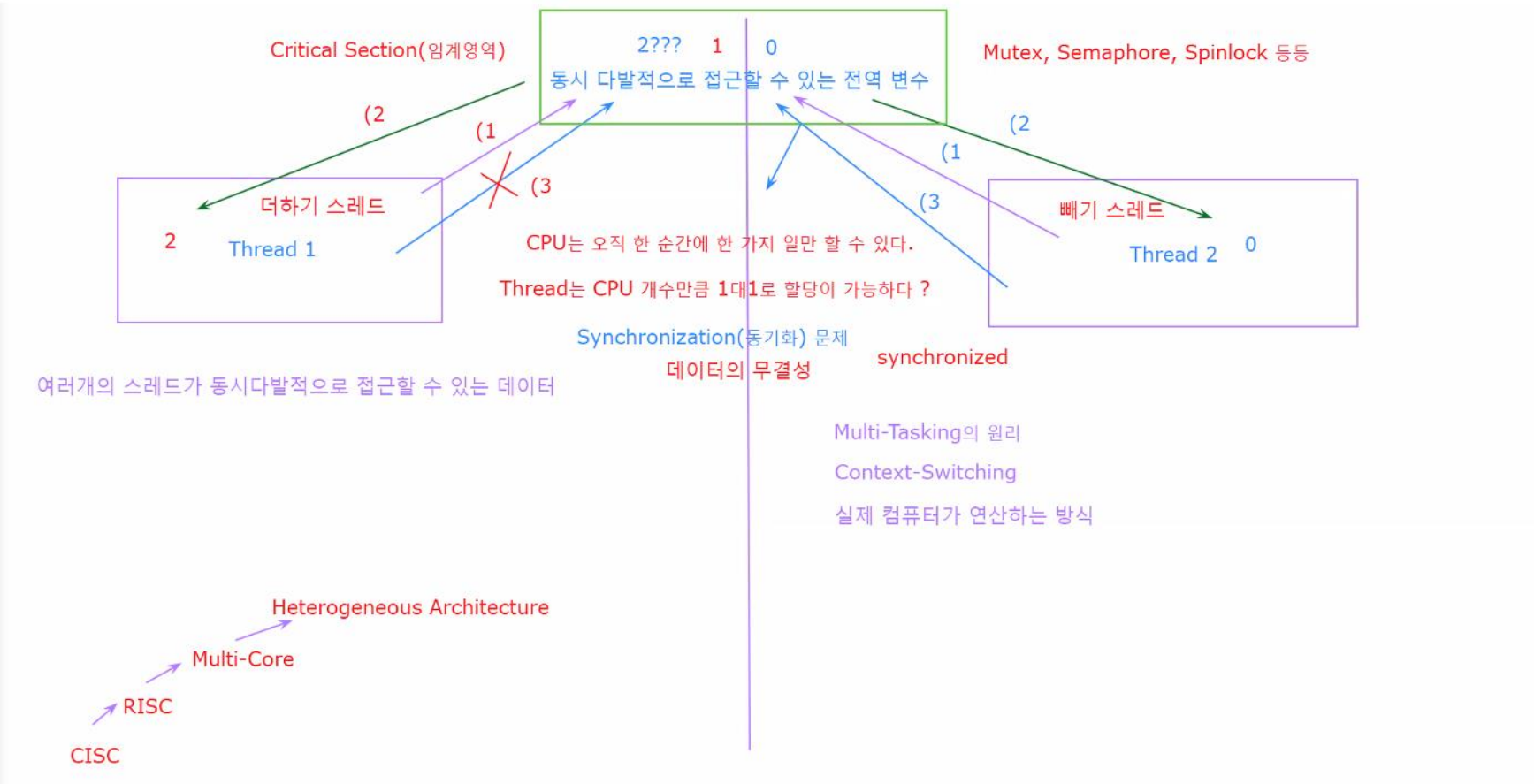
멀티 프로세스 :
서로 독립적. 하나의 프로세스에서 오류가 발생하더라도 다른 프로세스에 영향을 미치지 않음.

정리 :
프로세스는 운영체제로부터 작업을 할당받는 작업의 단위이고,
스레드는 프로세스가 할당 받은 자원을 이용하는 실행의 단위이다.

멀티쓰레드

장점
1.시스템자원을 보다 효율적으로 사용 2. 사용자에 대한 응답성이 향상 3. 작업이 분리되어 코드가 간결해진다.
ex)채팅프로그램에서 파일을 전송하면서 동시에 채팅을 할수 있다.

단점
1.동기화에 주의해야한다. 2.교착상태(DEAD-LOCK)이 발생하지않게 주의 3.각 쓰레드가 효율적으로 고르게 실행될수 있게 해야한다.(기아)



멀티 스레드에서 발생할수있는 문제
1) 1 전역변수에 접근 2) 1을 더해서 2가 된다 3) 재 접근과정에서 인터럽트 발생으로 작업중단
4) 두번째 스레드에서 전역변수에 접근하면서 값이 변경되고 문제발생 (Synchronization 문제)

이런 문제를 방지하기위해 Mutex,Semaphore,Spinlock 등이 임계구역에서 다른 스레드의 접근을 차단한다.

문맥교환 (context switching)

```
ex) 컨텍스트 스위칭 도입이 없는 상태
Thread 1
mov eax, 3    // eax = 3                    (1)
mov ebx, 4    // ebx = 4                    (4)
add eax, ebx  // eax = eax + ebx  ---> ???  (5) 7 + 4 = 11 ??????

Thread 2
mov eax, 7     // eax = 7                   (2)
mov ebx, 2     // ebx = 2                   (3)
add eax, ebx   // eax = eax + ebx

ex) 컨텍스트 스위칭을 도입한 상태
Thread 1
mov eax, 3    // eax = 3                    (1) ----> 제어권을 넘기기 전에 현재 하드웨어 레지스터 정보를 메모리에 저장함
                                                       (이 정보는 운영체제가 관리하고 있어서 찾을 수 있음)
mov ebx, 4    // ebx = 4                    (4) <--- 다시 돌아와서 이전에 저장한 정보를 메모리에서 찾아서 복원함
add eax, ebx  // eax = eax + ebx  ---> ???  (5) 3 + 4 = 7 (데이터의 무결성을 보장함)
Thread 2
mov eax, 7     // eax = 7                   (2)
mov ebx, 2     // ebx = 2                   (3) ----> 자신의 제어권이 역시 넘어가기 전에 하드웨어 레지스터 정보를 백업함
add eax, ebx   // eax = eax + ebx
```

스핀락 (문맥교환x)

짧은 시간 안에 진입할수 있는 경우 문맥 교환비용이 들지않아 효율을 높일 수있지만 반대로 작업이 오래걸리는 경우 다른 스레드에서 cpu를 양보하지 않기 때문에 오히려 cpu효율을 떨어뜨린다. (busy waiting)

뮤텍스 (문맥교환O)

acquire로 lock얻은 사람이 CS에진입한다 이때 available은 false가 되고 다른 작업이 못들어온다.
그리고 CS 나갈땐 release해서 avaliable을 True로 만들고 다시 CS에 들어갈때까지 available 체크할때 busy waiting한다.

Thread 사용

```java
import java.util.Random;
class Car implements Runnable {
    String name;
    private int sleepTime;
    private final static Random generator = new Random();

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

    @Override
    public void run() {
        try {

            Thread.sleep(sleepTime);
        } catch (InterruptedException e) {
            System.out.println("출력도 안 될 것이고 에러가 발생할 일이 없습니다!");
        }
        System.out.println(name + "이 경주를 완료하였습니다!");
    }
}
```

Runnable의 경우 run() 매서드에 대한 구현이 필요함

Random 클래스로 만든 객체에 nextInt() 매서드를 통해서도 랜덤값을 생성할 수 있다.

스레드를 돌릴땐 무조건 이 run() 부분을 구동시키게 되어있다.

catch()예외처리 : 인터럽트 등이 발생하는 경우에 작성하게 됨 에러를 유발할 수도 있다를 암시함

```java
public class ThreadTest {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Car( name: "Ferrari"));
        Thread t2 = new Thread(new Car( name: "Spyder 918"));
        Thread t3 = new Thread(new Car( name: "Maserati MC20"));

        t1.start();
        t2.start();
        t3.start();
    }
}
```

출력 값

```
Ferrari이 경주를 완료하였습니다!
Maserati MC20이 경주를 완료하였습니다!
Spyder 918이 경주를 완료하였습니다!
```

퀴즈는 못 풀었습니다.