

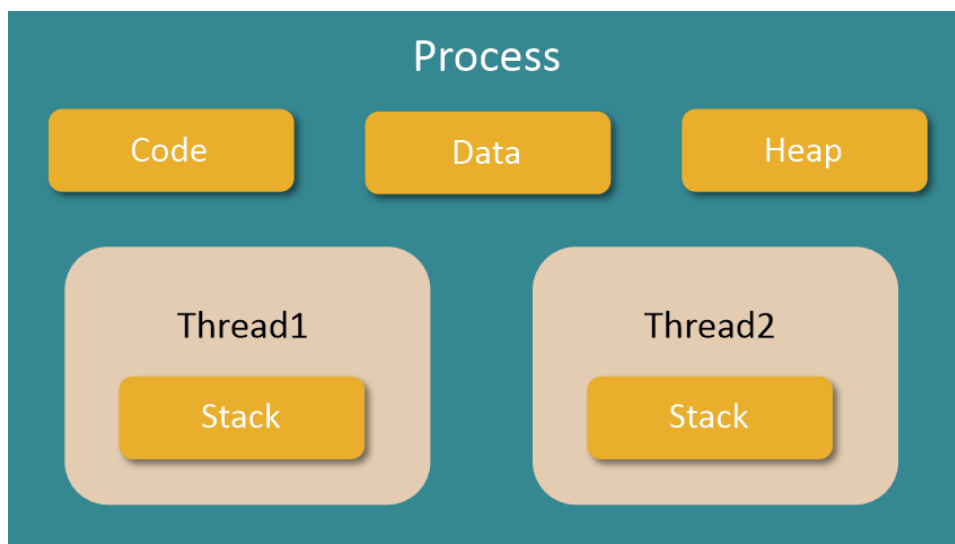


Thread

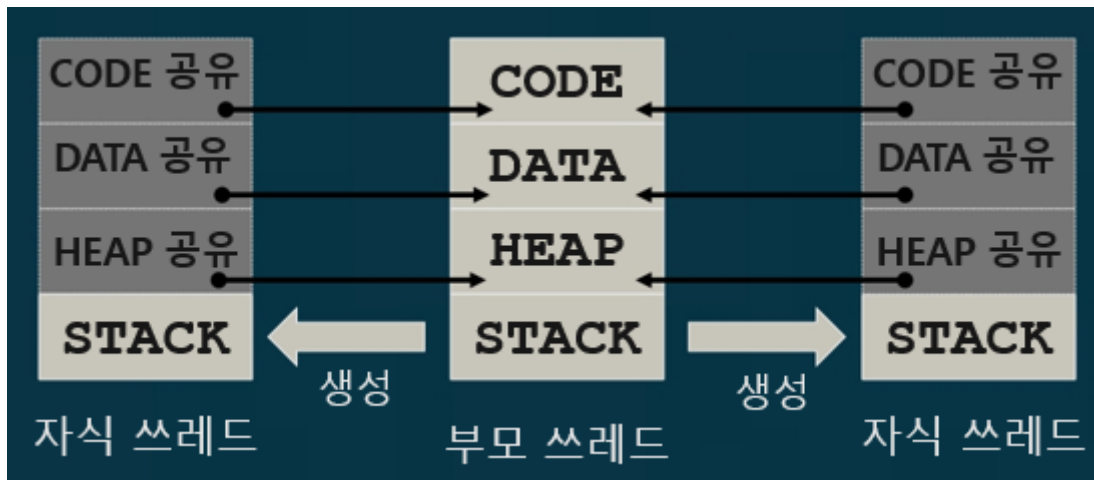
스레드



한 프로세스 내에서 동작되는 여러 실행의 흐름으로
프로세스 하나에 자원을 공유하면서 일련의 과정을 여러 개를 동시에 실행시킬
수 있는 것



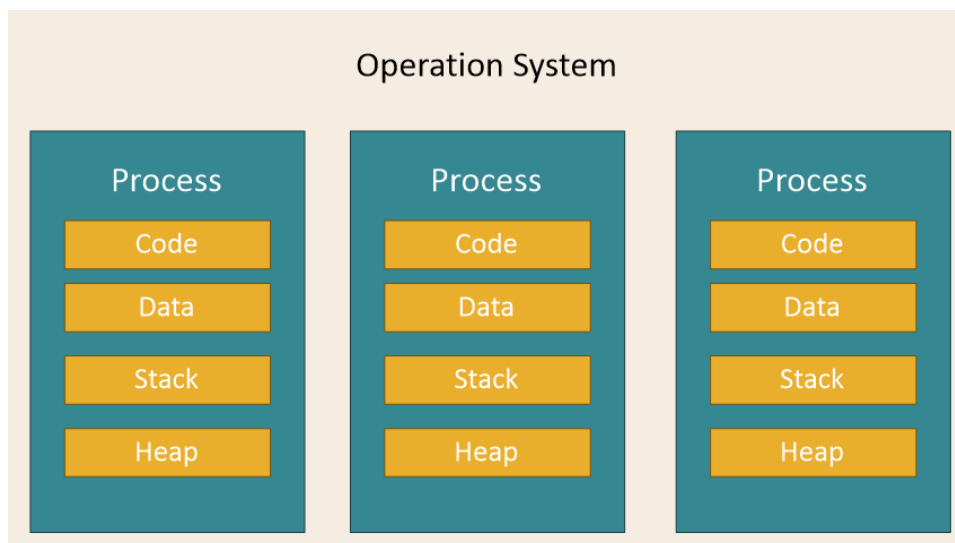
- 위 그림을 보면, 각 스레드는 Code, Data, Heap을 공유하고 Stack 만 별도로 가진다.
 - createThread 시스템 콜에 의한 부모-자식 스레드가 위의 그림과 같다고 보면 된다.
- ▼ 왜 Stack만 별도로 가질까?



모두 공유되면 똑같은 일밖에 할 수 없다. 공유가 되긴 하는데 똑같은 일을 하게된다면 의미가 없다. 그래서 **개인작업공간을 분리시킨 것이 스택이다.**

결국 스택은 각 스레드마다 독립적이기에 지역변수는 서로 볼 수 없고 수정할 수 없다.

멀티프로세스 VS 멀티스레드



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

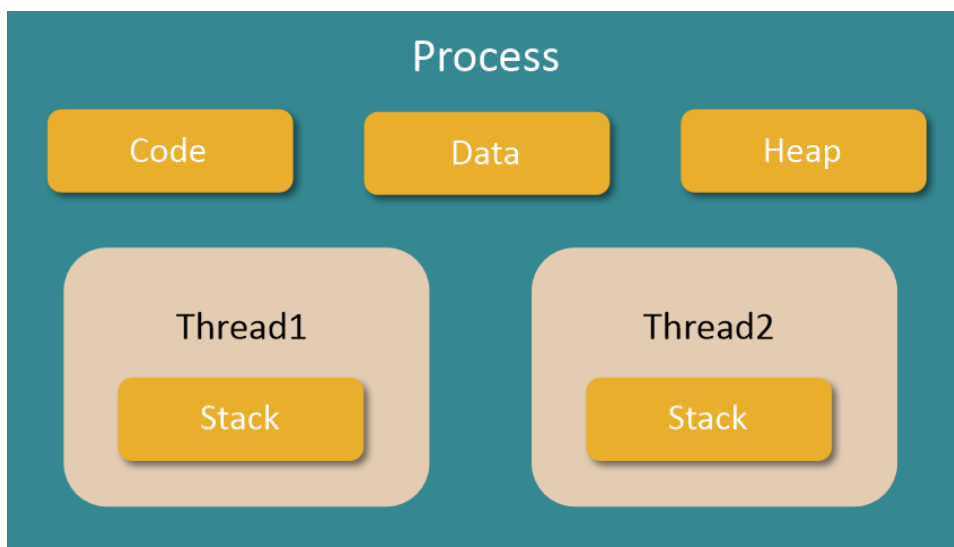
■ **장점** : 안전성이 높음 (독립된 구조기 때문에) — 여러 개의 자식 프로세스 중 하나에 문제가 발생하면 그 자식만 죽어서 영향이 확산 되지 않는다

■ **단점**

- Context Switching의 오버헤드: 캐쉬 메모리 초기화 등 무거운 작업이 진행되고 많은 시간이 소모되어 발생, IPC 통신의 무거움

▼ 🖱️ IPC가 왜 무겁지....?

운영체제마다 다 다른 API를 제공한다. 윈도우는 윈도우에 있는 IPC를 써야 하고, 리눅스는 리눅스에 있는 IPC를 써야 한다. 프로세스끼리 데이터를 주고받고 싶으면 소켓을 만들어서 소켓 IPC를 해야 한다. 그러면 같은 컴퓨터에 프로세스가 있지 않아도 된다. 다른 컴퓨터에 있는 프로세스끼리도 통신할 수 있다. 그런 엄청난 장점이 있지만, 오버헤드가 크고 느리다.



장점

- Context Switching 시 공유 메모리만큼 시간 손실이 줄어든다. (스택만 바꿔치기하면 끝!)
- 시스템 자원 소모 감소 - 메모리의 공유 덕분에 IPC 통신이 필요없어 데이터를 주고받는 것이 간단해진다.

단점

- 여러 개의 스레드를 이용하는 경우, 미묘한 시간차나 잘못된 변수 공유로 인해 오류 발생 가능
 - 🖱️ 스레드 간의 통신은 충돌 문제가 발생하지 않도록 동기화 문제의 해결이 필요
- 프로그램 디버깅이 어렵다.
- 단일 프로세스 시스템에서는 효과를 기대하기 어렵다.
- 스레드 하나가 죽으면 전체 스레드에 영향을 끼친다.(매우 치명적....)

java에서 스레드를 작성하는 방법은 2가지가 존재

👉 Thread 클래스 상속

‘클래스’이기 때문에 thread 상속을 받으면 다른 클래스 상속이 불가

▼ 코드

```
import java.util.Random;

public class MyThread extends Thread {

    private static final Random random = new Random();

    @Override
    public void run() {
        String threadName = Thread.currentThread().getName();
        System.out.println("- " + threadName + " has been started");
        int delay = 1000 + random.nextInt(4000);
        try {
            Thread.sleep(delay);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("- " + threadName + " has been ended (" + delay + "ms)");
    }
}
```

👉 Runnable 인터페이스 구현

더 정확히는 Runnable 인터페이스를 구현해서 thread 객체를 직접 만들어 사용

확장성이 더 중요한 클래스의 경우 runnable 방식을 채용하는 것이 적합 → 실제로도 많이 쓰임

▼ 코드

```
import java.util.Random;

public class MyRunnable implements Runnable {

    private static final Random random = new Random();

    @Override
    public void run() {
        String threadName = Thread.currentThread().getName();
        System.out.println("- " + threadName + " has been started");
    }
}
```

```

        int delay = 1000 + random.nextInt(4000);
        try {
            Thread.sleep(delay);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("- " + threadName + " has been ended (" + delay + "ms)");
    }
}

```

예제 : <https://inor.tistory.com/8?category=708183>

Runnable과 Thread의 실행

```

public class ThreadRunner {


    public static void main(String[] args) {
        // create thread objects
        Thread thread1 = new MyThread();
        thread1.setName("Thread #1");
        Thread thread2 = new MyThread();
        thread2.setName("Thread #2");

        // create runnable objects
        Runnable runnable1 = new MyRunnable();
        Runnable runnable2 = new MyRunnable();

        Thread thread3 = new Thread(runnable1); // Runnable 형 인자를 받는 생성자를 통해
        thread3.setName("Thread #3");           // 별도의 Thread 객체를 생성 후 start() 메소드 호출
        Thread thread4 = new Thread(runnable2);
        thread4.setName("Thread #4");

        // start all threads
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
    }
}

```

 스레드의 시작은 run()함수가 아닌 start() 함수!

▼ 그 이유

run() 메소드를 이용한다는 것은 main()의 콜 스택 하나만 이용하는 것으로 스레드 활용이 아니다. (그냥 스레드 객체의 run이라는 메소드를 호출하는 것 뿐이게 되는 것..)

start() 메소드를 호출하면, JVM은 알아서 스레드를 위한 콜 스택을 새로 만들어주고 context switching을 통해 스레드답게 동작하도록 해준다.

결론적으로, start()는 스레드가 작업을 실행하는데 필요한 콜 스택을 생성한 다음 run()을 호출해서 그 스택 안에 run()을 저장할 수 있도록 해준다.

JAVA에서의 멀티스레드 동기화

여러 스레드가 같은 프로세스 내의 자원을 공유하면서 작업할 때 서로의 작업이 다른 작업에 영향을 주므로 스레드의 동기화는 필수!



이때, 임계영역과 lock을 활용한다!

임계영역을 지정하고, 임계영역을 가지고 있는 lock을 단 하나의 스레드에게만 빌려주는 개념으로 이루어져있다.

따라서 임계구역 안에서 수행할 코드가 완료되면, lock을 반납해줘야 한다.

스레드 동기화 방법

- 임계 영역(critical section) : 공유 자원에 단 하나의 스레드만 접근하도록(하나의 프로세스에 속한 스레드만 가능)
- 뮤텍스(mutex) : 공유 자원에 단 하나의 스레드만 접근하도록(서로 다른 프로세스에 속한 스레드도 가능)
- 이벤트(event) : 특정한 사건 발생을 다른 스레드에게 알림

- 세마포어(semaphore) : 한정된 개수의 자원을 여러 스레드가 사용하려고 할 때 접근 제한
- 대기 가능 타이머(waitable timer) : 특정 시간이 되면 대기 중이던 스레드 깨움

synchronized 활용 - 임계영역 설정

synchronized를 활용해 임계영역을 설정할 수 있다.

서로 다른 두 객체가 동기화를 하지 않은 메소드를 같이 오버라이딩해서 이용하면, 두 스레드가 동시에 진행되므로 원하는 출력 값을 얻지 못한다.

이때 오버라이딩되는 부모 클래스의 메소드에 synchronized 키워드로 임계영역을 설정해주면 해결할 수 있다.

```
//synchronized : 스레드의 동기화. 공유 자원에 lock
public synchronized void saveMoney(int save){    // 입금
    int m = money;
    try{
        Thread.sleep(2000);    // 지연시간 2초
    } catch (Exception e){

    }
    money = m + save;
    System.out.println("입금 처리");
}

public synchronized void minusMoney(int minus){    // 출금
    int m = money;
    try{
        Thread.sleep(3000);    // 지연시간 3초
    } catch (Exception e){

    }
    money = m - minus;
    System.out.println("출금 완료");
}
```

#wait()과 notify() 활용

스레드가 서로 협력관계일 경우에는 무작정 대기시키는 것으로 올바르게 실행되지 않기 때문에 사용한다.

- wait() : 스레드가 lock을 가지고 있으면, lock 권한을 반납하고 대기하게 만듦
- notify() : 대기 상태인 스레드에게 다시 lock 권한을 부여하고 수행하게 만듦

이 두 메소드는 동기화 된 영역(임계 영역)내에서 사용되어야 한다.

동기화 처리한 메소드들이 반복문에서 활용된다면, 의도한대로 결과가 나오지 않는다. 이때 wait()과 notify()를 try-catch 문에서 적절히 활용해 해결할 수 있다.

```
/**
 * 스레드 동기화 중 협력관계 처리작업 : wait() notify()
 * 스레드 간 협력 작업 강화
 */

public synchronized void makeBread(){
    if (breadCount >= 10){
        try {
            System.out.println("빵 생산 초과");
            wait();    // Thread를 Not Runnable 상태로 전환
        } catch (Exception e) {

        }
    }
    breadCount++;    // 빵 생산
    System.out.println("빵을 만들. 총 " + breadCount + "개");
    notify();    // Thread를 Runnable 상태로 전환
}

public synchronized void eatBread(){
    if (breadCount < 1){
        try {
            System.out.println("빵이 없어 기다림");
            wait();
        } catch (Exception e) {

        }
    }
    breadCount--;
    System.out.println("빵을 먹음. 총 " + breadCount + "개");
    notify();
}
```

조건 만족 안할 시 wait(), 만족 시 notify()를 받아 수행한다.

<https://tecoble.techcourse.co.kr/post/2021-10-23-java-synchronize/>