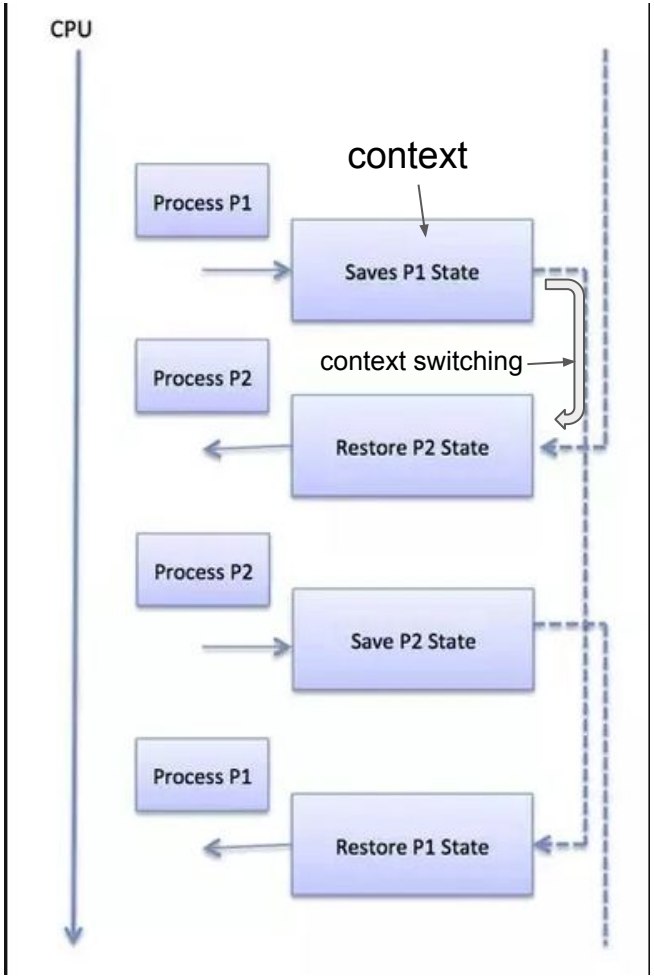
Context Switching

멀티프로세스 환경에서 CPU가 어떤 하나의 프로세스를 실행하고 있는 상태에서 인터럽트,이벤트 요청에 의해 다음 우선 순위의 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 작업**을 Context Switch(Context Switching)라고 한다.
질문에 대한 답변은 이정도로 하고 좀 더 명확하게 이해해본다.

* Context Switching을 문맥 교환으로 번역하지 말자.



Mutex 와 Spinlock

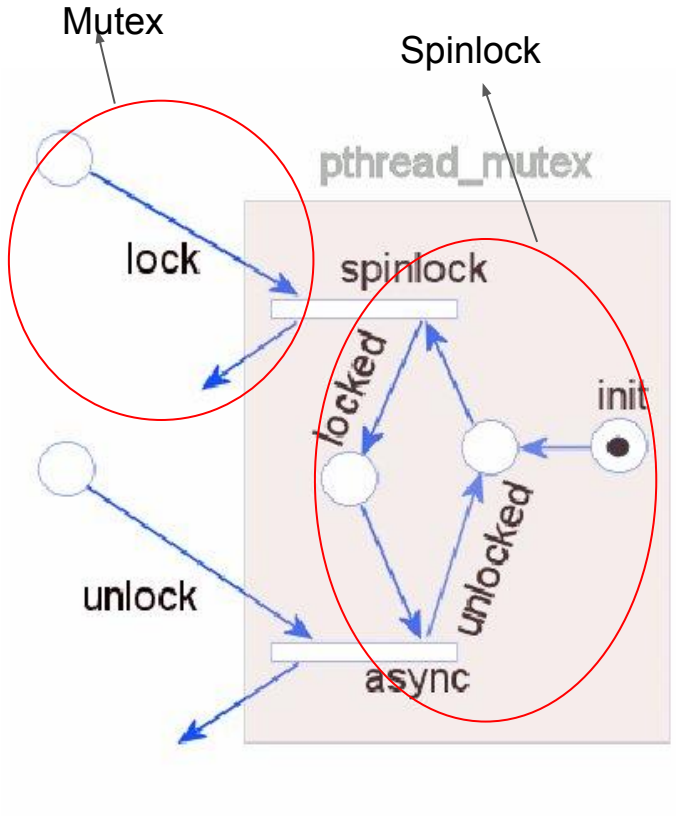
뮤텍스의 경우, 자원에 이미 락이 걸려 있을 경우 락이 풀릴 때까지 기다리며 컨텍스트 스위칭을 실행한다.

즉, 다른 병렬적인 태스크를 처리하기 위해 CPU를 양보할 수 있다는 것이며 이는 자원을 얻기 위해 오랜 시간을 기다려야 할 것이 예상될 때 다른 작업을 동시에 진행할 수 있다는 것이다. 하지만 이는 자원이 단시간 내로 얻을 수 있게 될 경우 컨텍스트 스위칭에 더 큰 자원을 낭비하게 될 수 있다는 문제가 있다.

스핀 락의 경우에는 이름에서부터 알 수 있듯이, 자원에 락이 걸려 있을 경우 이를 얻을 때까지 무한 루프를 돌면서 다른 태스크에 CPU를 양보하지 않는 것이다. 자원이 단시간 내로 얻을 수 있게 된다면 컨텍스트 스위칭 비용이 들지 않으므로 효율을 높일 수 있지만, 그 반대의 경우 다른 태스크에 CPU를 양보하지 않으므로 오히려 CPU 효율을 떨어뜨릴 수 있는 문제가 있다.

경우에 따라선 spinlock이 더 효율적인 경우가 있다.

단순작업 -- 그렇다면 단순 작업의 기준은 어디까지인가...



```java
public class PerformanceTest2 {
    final static int ZERO = 0;
    final static int END = 1000000000;
    final static int START = 1;
    final static double COEFFICIENT = Math.pow(10, -15);
    final static double DEG2RAD = 180.0;
    final static double test = Math.PI / DEG2RAD;
    public static void main(String[] args) {
        double sum = ZERO;

        PerformanceUtil.performanceCheckStart();

        for(int i = START; i <= END; i++) {
            sum += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
        }

        PerformanceUtil.performanceCheckEnd();

        System.out.println("sum = " + sum);

        PerformanceUtil.printPerformance();
    }
}
```

공통 변수로 선언하고 값을 계산하면 걸리는 시간이 빠르게 나타났다.
그렇다면? 공통으로 묶을 수 있는 것들을 묶으면 그만큼 시간이 더 단축되는 것 인가요?
(Math.PI/DEG2RAD) 도 test로 묶어서 결과를 도출해내면 2초정도 더 단축되었습니다.

어제 문제풀고있는 도중이라 제대로 듣질 못했는데 static을 너무 자주 사용하는것이 좋지 않다 말씀하셨던거 같은데 이유를 모르겠습니다.

```java
Thread[] thr = new Thread[5];

for(int i = 0; i < 5; i++) {
    thr[i] = new Thread(new AccelThread(START, END, maxThreadNum: 5, i));
}

public AccelThread(int start, int end, int maxThreadNum, int id) {
    super(start, end, maxThreadNum);

    int total = end - start + 1;
    int threadPerData = total / maxThreadNum;

    localStart = id * threadPerData + 1;
    localEnd = localStart + threadPerData - 1;
    threadId = id;
}
```

Thread 5개 생성
for문으로 각 Thread에 AccelThread 생성자 입력 및 호출
thr[0]
start = 1
end = 10억
maxThreadNum =5
i =0

total = 10억 -1+1 = 10억
threadPerData =10억/5 = 2억

localstart = 0*2억 +1 =1
localend = 1+ 2억 -1 = 2억
threadId = id =i ->0

식으로 0~4까지의 5개의 thread가 생성된다.

각 스레드의 계산값을 더하기. 까지가... 결과값일듯 하지만
프로그래밍을 못하겠습니다..