프로세스와 스레드

💡프로세스(Process)

> 📌 실행파일(program) 메모리에 적재되어 CPU를 할당받아 실행되는 것을 프로세스라고 한다.

- 프로세스란 실행 중인 프로그램을 뜻한다. 즉, 실행파일 형태로 존재하던 program이 메모리에 적재되어 운영체제에 의해서 CPU에 실행(연산)되는 것을 프로세스라고 한다.

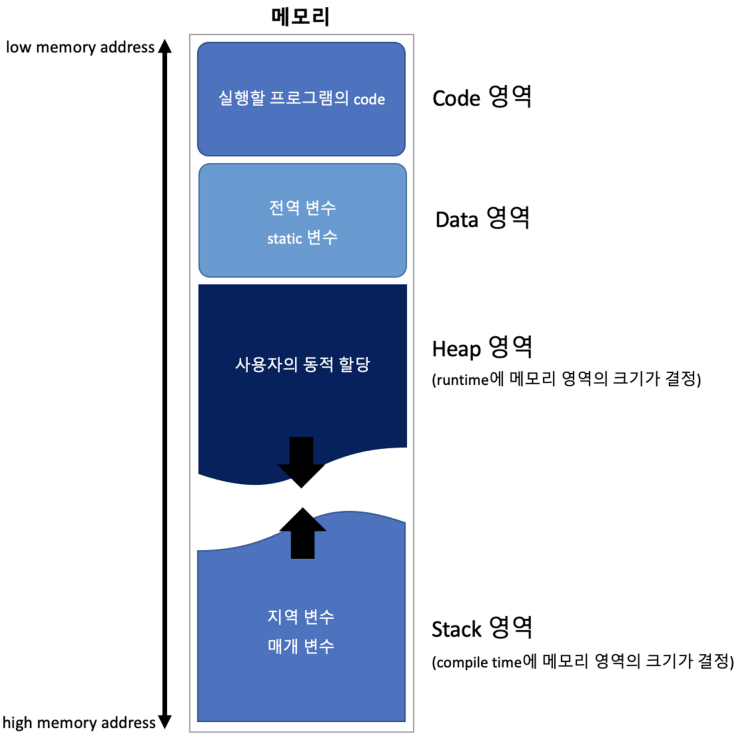
정적 프로그램 (Static Program)

program은 윈도우의 `*.exe` 파일이나 Mac의 `*.dmg` 파일과 같은 **컴퓨터에서 실행할 수 있는 파일**을 통칭한다. 단, 아직 파일을 실행하지 않은 상태이기 때문에 정적 프로그램, 줄여서 프로그램이라 칭한다. 어떠한 프로그램을 개발하기 위해선 자바나 C언어와 같은 언어를 이용해 코드를 작성하여 완성된다. 즉, program은 쉽게 말해서 그냥 **코드 덩어리**라고 할 수 있다. 이러한 program을 실행시킨 것이 바로 process이다.

Memory에 적재

메모리는 **CPU가 접근할 수 있는 컴퓨터 내부의 기억장치**이다. program이 CPU에서 실행 되려면 반드시 메모리에 먼저 올라가야 한다.

프로세스에 할당되는 메모리 공간은 `Code`, `Data`, `Stack`, `Heap` 4개의 영역으로 이루어져 있으며, **각 프로세스마다 독립적으로 할당**받는다.



Code 영역	실행할 프로그램의 코드가 CPU가 해석 가능한 기계어 형태로 저장되어 있는 메모리 영역
Data 영역	프로그램의 전역 변수와 static 변수가 저장되는 메모리 영역
Heap 영역	프로그래머가 직접 공간을 할당(malloc)/해제(free) 하는 메모리 영역. 생성자, 인스턴스와 같은 동적으로 할당되는 데이터들을 위해 존재하는 공간이다.
Stack 영역	함수 호출 시 생성되는 지역 변수와 매개 변수가 저장되는 임시 메모리 영역. 함수의 호출과 함께 할당되며, 함수의 호출이 완료되면 소멸한다. 만일 stack 영역을 초과하면 stack overflow 에러가 발생한다.

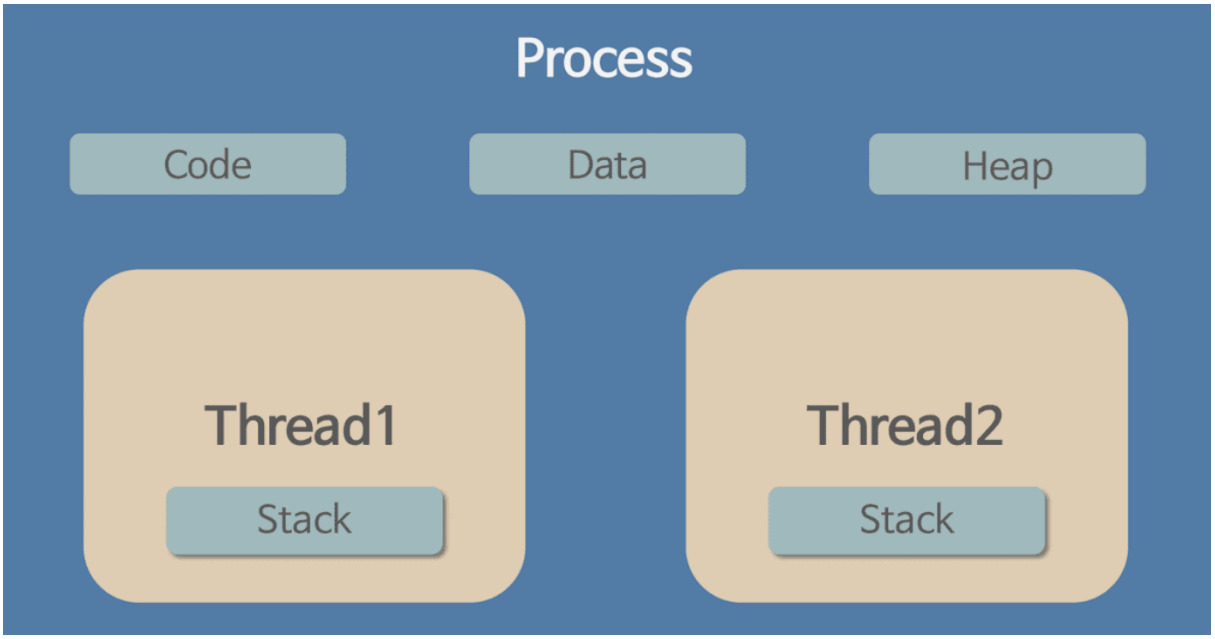
💡스레드(Thread)

> 📌 스레드란 하나의 **프로세스 내에서 동시에 진행되는 작업의 갈래, 또는 흐름의 단위**를 말한다.

- 하나의 프로세스 안에 여러개의 스레드가 존재한다. 스레드 수가 많을수록 프로그램 성능 또한 올라간다.
- 프로세스를 생성하면 기본적으로 하나의 main 스레드가 생성된다.

스레드의 자원 공유

프로세스가 자신만의 고유 공간과 자원을 할당 받아 사용하는데 반해, **스레드는 다른 스레드와 공간, 자원을 공유하면서 사용**한다.



프로세스의 4가지 메모리 영역(`Code`, `Data`, `Heap`, `Stack`) 중 스레드는 `Stack`만 할당 받아 복사하고 `Code`, `Data`, `Heap`은 프로세스 내의 다른 스레드들과 공유된다. 따라서 각각의 스레드는 별도의 `stack`을 가지고 있지만 `heap` 메모리는 고유하기 때문에 서로 다른 스레드에서 가져와 읽고 쓸 수 있게 된다.

이렇게 구성하는 이유는 하나의 프로세스를 다수의 실행 단위인 스레드로 구분하여 자원을 공유하고, 자원의 생성과 관리의 중복성을 최소화하여 수행 능력을 올리기 위해서이다.

스레드가 Stack 영역만 따로 할당 받는 이유

- 독립적인 동작 수행을 위해서!
- 독립적으로 함수를 호출해야 한다.
- `stack`은 함수 호출 시 전달되는 매개변수, 되돌아갈 주소값, 지역 변수 등을 저장하는 메모리 공간이다.
- 즉, 독립적인 `stack`을 가졌다는 것은 독립적인 함수 호출이 가능하다는 뜻이다.
- `stack`을 할당받는 스레드는 독립적인 실행 흐름을 가질 수 있다.

💡 멀티 프로세스(Multi process)

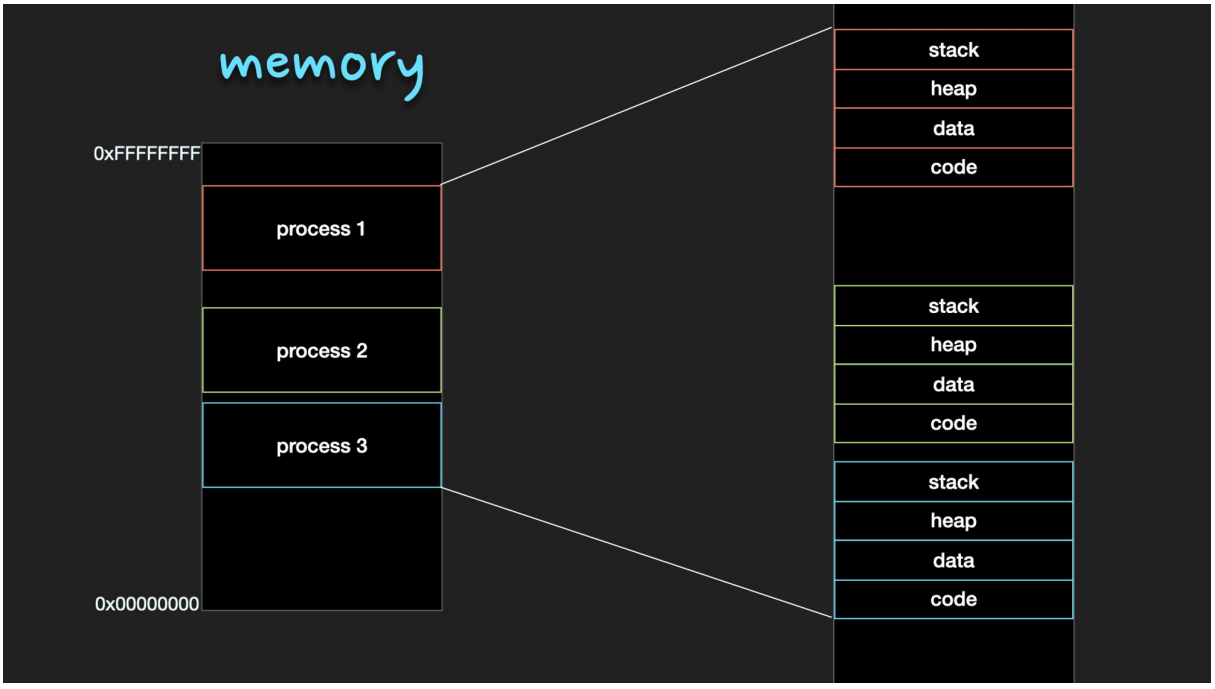
📌 2개 이상의 프로세스가 동시에 실행되는 것을 말한다. 동시에 라는 것은 **동시성(concurrency)**와 **병렬성(parallelism)** 두 가지를 의미한다.

- 동시성: CPU core가 1개일 때, 여러 프로세스를 짧은 시간동안 번갈아 가면서 연산을 하게 되는 시분할 시스템(time sharing system)으로 실행되는 것
- 병렬성: CPU core가 여러개일 때, 각각의 core가 각각의 프로세스를 연산함으로써 프로세스가 동시에 실행되는 것

동시성(concurrency)	병렬성(parallelism)
Single Core	Multi Core
동시에 실행되는 것처럼 보인다.	실제로 동시에 여러 작업이 처리된다.

멀티 스레드란 2개 이상의 프로세스가 동시에 실행되는 것을 말하는데, 이 때 프로세스들은 CPU와 메모리를 공유하게 된다. 메모리의 경우에는 여러 프로세스들이 각각의 독립된 메모리 영역을 동시에 할당 받는다.

반면에 하나의 CPU는 매 순간 하나의 프로세스만 연산할 수 있다. 하지만 CPU의 처리 속도가 워낙 빨라서 수 ms 이내의 짧은 시간동안 여러 프로세스들이 CPU에서 번갈아 실행되기 때문에 사용자 입장에서는 여러 프로그램이 동시에 실행되는 것처럼 보인다. 이처럼 CPU의 작업 시간을 여러 프로세스들이 조금씩 나누어 쓰는 시스템을 **시분할 시스템(time sharing system)**이라고 부른다.



여러 프로세스가 동시에 메모리에 적재된 경우, 서로 다른 프로세스의 영역을 침범하지 않도록 각 프로세스가 자신의 메모리에 영역에만 접근하도록 운영체제가 관리해준다.

멀티프로세스의 장점 vs 단점

- 장점: 안전성 (메모리 침범 문제를 OS 차원에서 해결)
- 단점: 각각 독립된 메모리 영역을 가지고 있기 때문에 작업량이 많을수록 오버헤드가 발생한다. 또한 **Context Swtiching**으로 인한 성능 저하가 발생한다.

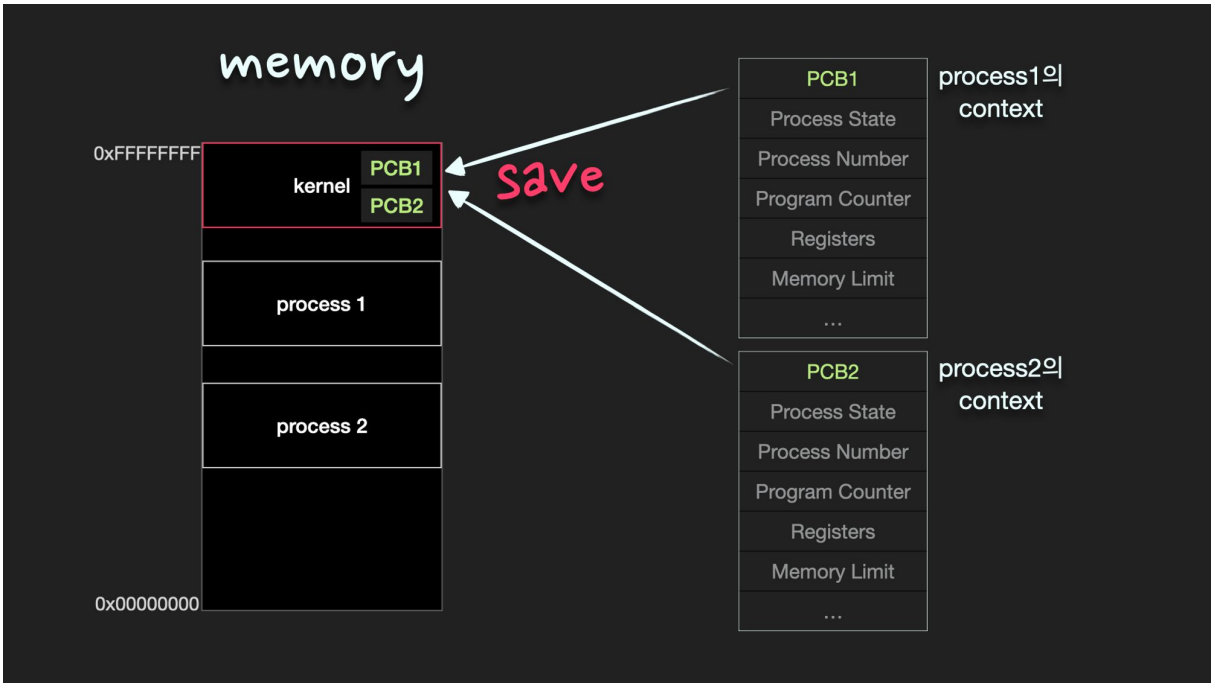
Context

시분할 시스템에서는 한 프로세스가 매우 짧은 시간동안 CPU를 점유하여 일정 부분의 명령을 수행하고, 다른 프로세스에게 넘긴다. 그 후 차례가 되면 다시 CPU를 점유하여 명령을 수행한다. 따라서 이전에 어디까지 명령을 수행했고, register에는 어떤 값이 저장되어 있었는지에 대한 정보가 필요하게 된다. 프로세스가 현재 어떤 상태로 수행되고 있는지에 대한 총체적인 정보가 바로 conext이다. context 정보들은 PCB(Process Control Block)에 저장된다.

💡PCB(Process Control Block)

> 📌 운영체제에서 프로세스를 관리하기 위해 해당 프로세스의 상태 정보를 담고 있는 자료구조를 말한다.

- 한 PCB 안에는 한 프로세스의 정보가 담긴다.
- PCB에는 프로세스의 중요한 정보가 포함되어 있기 때문에 일반 사용자가 접근하지 못하도록 보호된 메모리 영역 안에 저장된다.
- 일부 운영체제에서 PCB는 커널 스택에 위치한다. 해당 메모리 영역을 보호를 받으면서도 비교적 접근하기가 편리하다.
- 프로그램 실행 → 프로세스 생성 → 프로세스 주소 공간에 (코드, 데이터, 스택) 생성 → 이 프로세스의 메타데이터들이 PCB에 저장



PCB에는 일반적으로 다음과 같은 정보가 포함된다.

Process State	new, running, waiting, halted 등의 프로세스 상태들
Process Number	해당 process의 number
Program counter(PC)	해당 process가 다음에 실행할 명령어의 주소를 가리킨다.
Registers	컴퓨터구조에 따라 다양한 수와 유형을 가진 register 값들

Memory limits	base registers, limit register, page table 또는 segment table 등 운영체제에서 사용하는 메모리 관리 시스템에 대한 정보

PCB는 왜 필요할까?

> **CPU에서는 프로세스의 상태에 따라 교체작업**이 이루어진다. (interrupt가 발생해서 할당받은 프로세스가 waiting 상태가 되고 다른 프로세스를 running으로 바꾸어 올릴 때)

> 이 때, **앞으로 다시 수행할 대기 중인 프로세스에 관한 저장 값을 PCB에 저장**해 두는 것이다.

PCB는 어떻게 관리?

> LinkedList 방식으로 관리된다.

> PCB List Head에 PCB들이 생성될 때마다 붙게 된다. 주소값으로 연결이 이루어져 있는 연결리스트이기 때문에 삽입 삭제가 용이하다.

> 즉, 프로세스가 생성되면 해당 PCB가 생성되고 프로세스 완료시 제거된다.

이렇게 수행 중인 프로세스를 변경할 때, CPU의 레지스터 정보가 변경되는 것을 **Context Switching**이라고 한다.

💡Context Swtiching

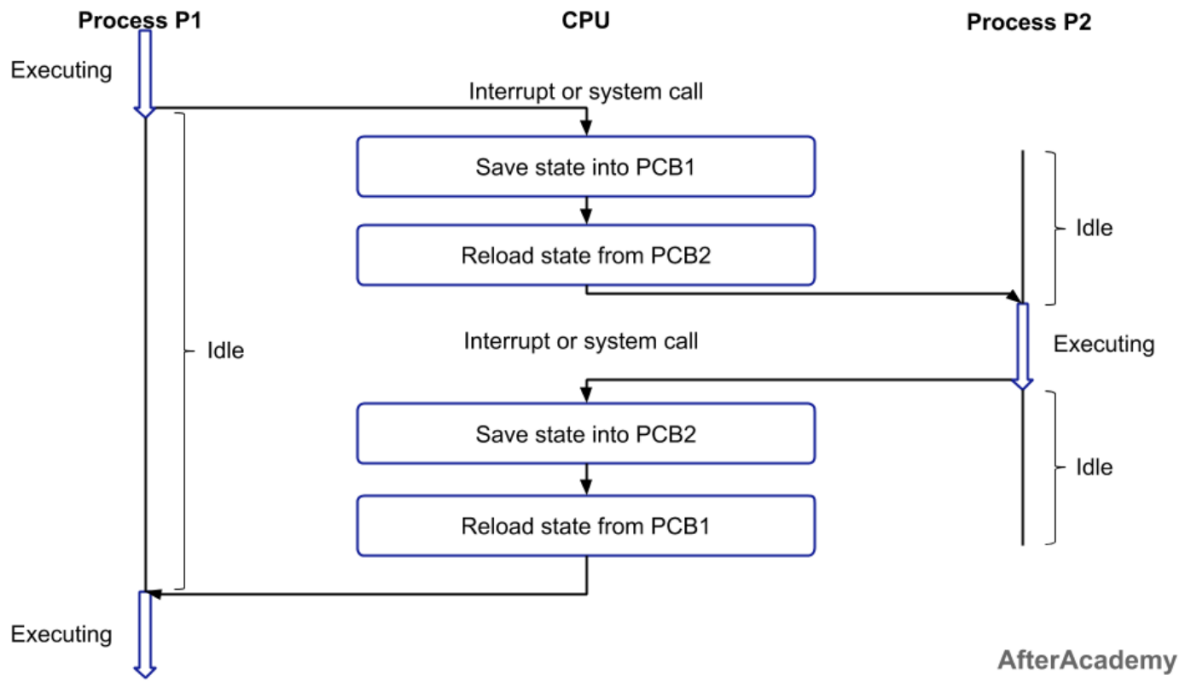
📌 한 프로세스에서 다른 프로세스로 CPU 제어권을 넘겨주는 것.

- 보통 인터럽트가 발생하거나, 실행 중인 CPU 사용 허가시간을 모두 소모하거나, 입출력을 위해 대기해야 하는 경우에 Context Switching이 발생한다.
- 이 때 이전의 프로세스 상태를 PCB에 저장하여 보관하고 새로운 프로세스의 PCB를 읽어서 보관된 상태를 복구하는 작업이 이루어진다.
- 컨텍스트 스위칭의 주체는 스케줄러이다.

Context Switching이 필요한 이유

CPU는 한 번에 하나의 프로세스만 실행할 수 있으므로 여러 개의 프로세스를 번갈아가면서 실행하여 CPU 활용률을 높이기 위해서 컨텍스트 스위칭이 필요하다.

Context Switching 과정

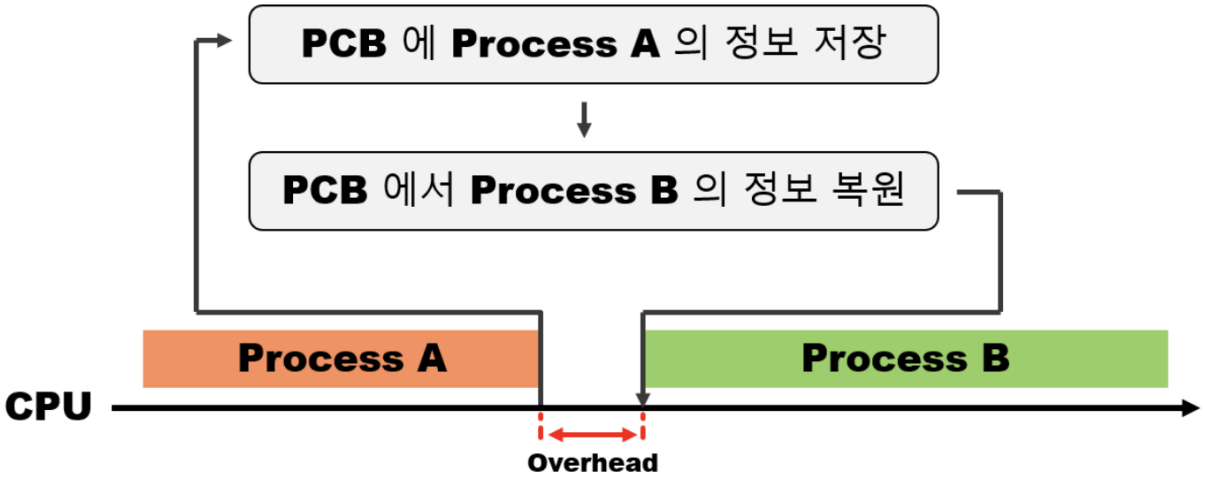


1. CPU는 Process P1을 실행한다 (Executing)
2. 일정 시간이 지나 Interrupt 또는 system call이 발생한다. (CPU는 idle 상태)
3. 현재 실행 중인 Process P1의 상태를 PCB1에 저장한다. (Save state into PCB1)
4. 다음으로 실행할 Process P2를 선택한다. (CPU 스케줄링)
5. Process P2의 상태를 PCB2에서 불러온다. (Reload state from PCB2)
6. CPU는 Process P2를 실행한다. (Executing)
7. 일정 시간이 지나 Interrupt 또는 system call이 발생한다. (CPU는 idle 상태)
8. 현재 실행 중인 Process P2의 상태를 PCB2에 저장한다. (Save state into PCB2)
9. 다시 Process P1을 실행할 차례가 된다. (CPU 스케줄링)
10. Process P1의 상태를 PCB1에서 불러온다. (Reload state from PCB1)
11. CPU는 Process P1을 중간 시점 부터 실행한다. (Executing)

Context Switching Overhead

컨텍스트 스위칭 과정은 사용자로금 빠른 반응성과 동시성을 제공하지만, 실행되는 프로세스의 변경 상태에서 프로세스의 상태, 레지스터 값 등이 저장되고 불러오는 등의 작업이 수행되기 때문에 시스템에 많은 부단을 주게 된다.

위의 컨텍스트 스위칭 과정 그림에서 P1이 Execute에서 idle이 될 때 P2가 바로 Execute가 되지 않고 idle 상태에 조금 있다가 Execute가 된다. 이 간극이 바로 Context Switching Overhead이다.



Context Switching Overhead를 유발하는 행위

- PCB 저장 및 복원 비용
- CPU 캐시 메모리 무효화에 따른 비용
- 프로세스 스케줄링 비용

> 컨텍스트 스위칭 과정에서 PCB를 저장하고 복원하는데 비용이 발생하며, 프로세스 자체가 교체되는 것이니 CPU의 캐시 메모리에 저장된 데이터가 무효화가 된다. 이 과정에서는 메모리 접근 시간이 늘어나고, 성능 저하가 발생할 수 있다. 또한 CPU 스케줄링 알고리즘에 따라 프로세스를 선택하는 비용도 만만치 않다.

컨텍스트 스위칭은 꼭 프로세스 뿐만 아니라 여러 개의 스레드들 끼리도 스위칭이 일어난다. 보통 멀티 스레드라고 하면 여러개의 스레드가 동시에 돌아가니 프로그램 성능이 무조건 상승할 것이라 생각하지만, 그렇지 않다. 컨텍스트 스위칭 오버헤드라는 변수 때문에 스레드 교체 과정에서 과하게 오버헤드가 발생하면 오히려 멀티 스레드가 싱글 스레드보다 성능이 떨어지는 현상이 나타날 수 있다.

Overhead를 감수해야 하는 상황

> 프로세스를 수행하다가 입출력 이벤트가 발생해서 대기 상태로 전환시킨다.
> 이때, CPU를 그냥 놀게 놔두는 것보다 다른 프로세스를 수행시키는 것이 효율적이다.

> 즉, CPU에 계속 프로세스를 수행시키도록 하기 위해서 다른 프로세스를 실행시키고 Context Switching 하는 것이다.
>
> CPU가 놀지 않도록 만들고, 사용자에게 빠르게 일처리를 제공해주기 위한 것이다.

💡 멀티 스레드(Multi Thread)

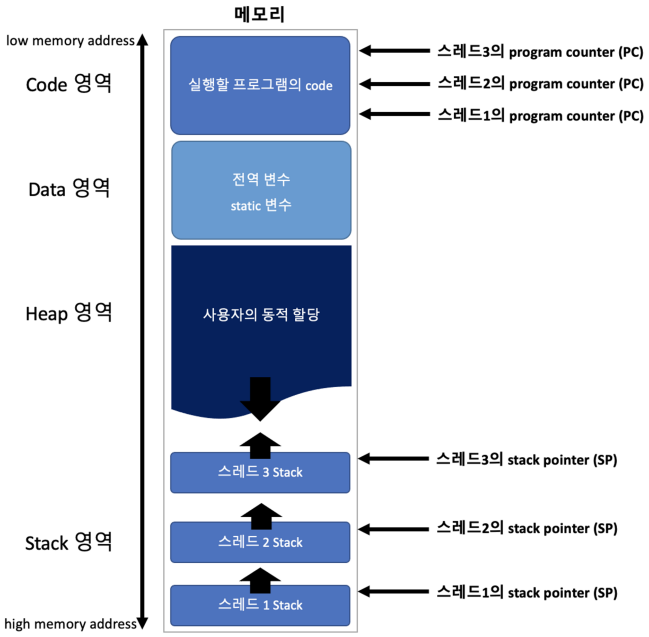
📌 하나의 프로세스가 동시에 여러 개의 일을 수행할 수 있도록 해주는 것

- 하나의 프로세스에서 여러 작업을 병렬로 처리하기 위해 멀티 스레드를 사용한다.
- 한 프로세스 내에 여러 개의 스레드가 있고, 각 스레드들은 `Stack` 메모리를 제외한 나머지 영역을 공유하게 된다.

Stack memory & PC register

스레드가 함수를 호출하기 위해서는 인자 전달, Return address 저장, 지역변수 저장 등을 위한 독립적인 stack memory 공간을 필요로 한다. 결과적으로 스레드는 프로세스로부터 stack memory 영역은 독립적으로 할당받고, code, data, heap 영역은 공유하는 형태를 갖게 된다.

또한 멀티 스레드에서는 각가의 스레드마다 PC register를 가지고 있어야 한다. 그 이유는 한 프로세스 내에서도 스레드끼리 context switch가 일어나는데, PC register에 code address가 저장되어 있어야 이어서 실행할 수 있기 때문이다.



멀티 스레드의 장점 vs 단점

- 장점: 독립적인 프로세스에 비해 공유 메모리만큼의 시간, 자원 손실이 감소한다. 전역 변수와 정적 변수에 대한 자료를 공유할 수 있다.
- 단점: 안전성 문제. 하나의 스레드가 데이터 공간을 망가뜨리면 모든 스레드가 작동 불능 상태가 된다.
  - 멀티스레드의 안정성에 대한 단점은 **Critical Section** 기법을 통해 대비한다.

> 하나의 스레드가 공유 데이터 값을 변경하는 시점에 다른 스레드가 그 값을 읽으려 할 때 발생하는 문제를 해결하기 위한 동기화 과정

> 상호 배제, 진행, 한정된 대기를 충족해야 한다.

💡 멀티 프로세스 vs 멀티 스레드

멀티 프로세스	멀티 스레드
많은 메모리 공간 / CPU 시간 차지	적은 메모리 공간 / CPU 시간 차지
Context Switching이 느림	Context Switching이 빠름
하나의 프로세스가 죽더라도 다른 프로세스에 영향을 주지 않아 안정성이 높음	동기화 문제와 하나의 스레드 장애로 전체 스레드가 종료될 위험 존재

출처:

https://www.nossi.dev/d3aca697-4795-4886-ba3d-c9f1b0526652

https://inpa.tistory.com/entry/👧-프로세스-⚔️-쓰레드-차이