프로세스와 스레드

프로세스 (Process)

컴퓨터에서 실행되고 있는 프로그램

CPU 스케줄링의 대상이 되는 작업 (Task)이라는 용어와 거의 같은 의미로 사용됨

스레드 (Thread)

프로세스 내 작업의 흐름

프로그램이 메모리에 올라가면 프로세스가 되는 인스턴스화가 일어남

이후 OS 의 CPU 스케줄러에 따라 CPU 가 프로세스를 실행함

프로세스와 컴파일 과정

프로세스는 프로그램으로부터 인스턴스화 된 것을 말함

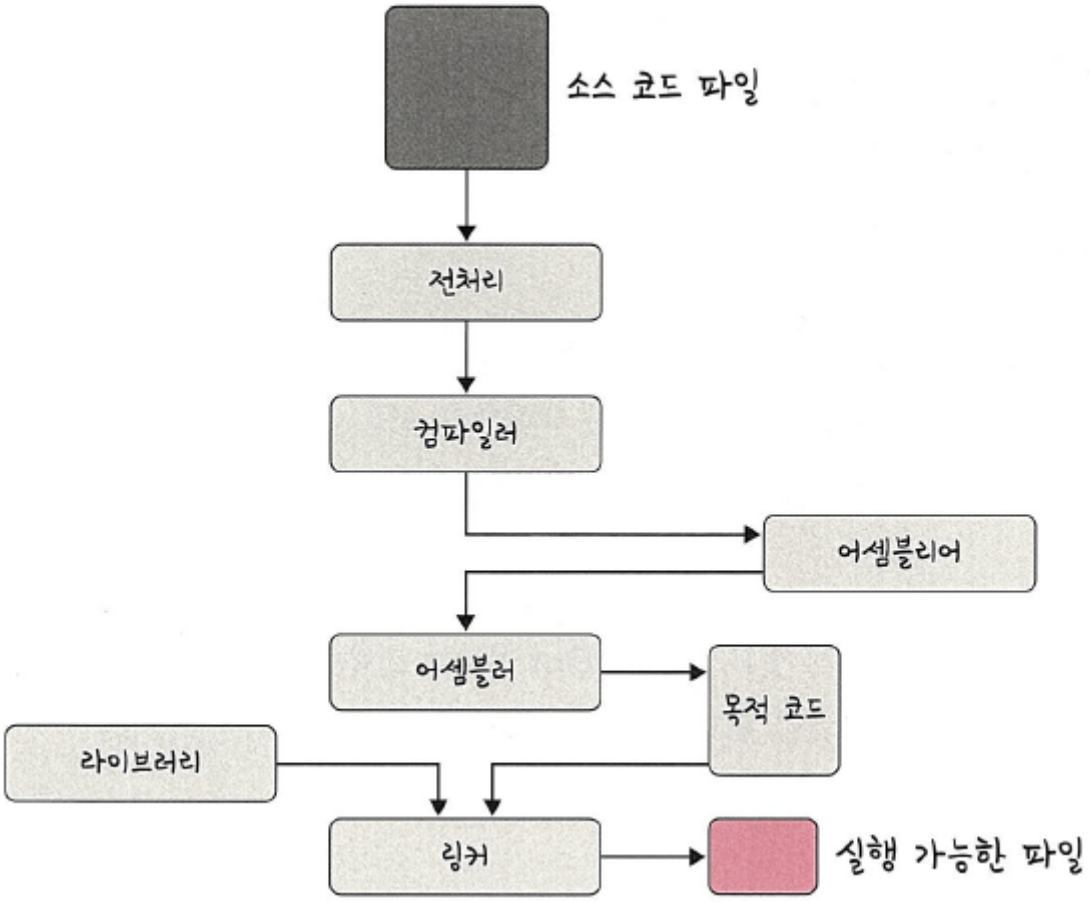
ex) 프로그램은 구글 크롬 프로그램(chrome.exe)과 같은 실행 파일. 이를 실행하면 구글 크롬 프로세스가 시작되는 것

프로그램은 컴파일러가 컴파일 과정을 거쳐 컴퓨터가 이해할 수 있는 기계어로 번역되어 실행 가능한 파일이 되는 것

컴파일 과정

C 언어를 기준으로 작성되었음

인터프리터 언어와는 다름 (별도의 컴파일 과정 없이 한번에 한줄씩 읽어들여 실행)



전처리

소스코드의 주석을 제거하고 #include 등 헤더 파일을 병합하여 매크로를 치환

컴파일러

오류 처리, 코드 최적화 작업 수행을 하며 어셈블리어로 변환

어셈블러

어셈블리어는 목적 코드 (Object Code)로 변환됨

확장자는 OS 마다 다름 (ex. 리눅스: .o)

링커

프로그램 내에 있는 라이브러리 함수 또는 다른 파일들과 목적 코드를 결합하여 실행 파일을 만듦

실행 파일의 확장자는 .exe 또는 .out 이라는 확장자를 가짐

정적 라이브러리와 동적 라이브러리

라이브러리는 정적과 동적으로 나뉨

- **정적 라이브러리**

  프로그램 빌드 시 라이브러리가 제공하는 모든 코드를 실행 파일에 넣는 방식

  외부 의존도가 낮다는 장점이 있으나, 코드 중복 등 메모리 효율성이 떨어지는 단점이 있음

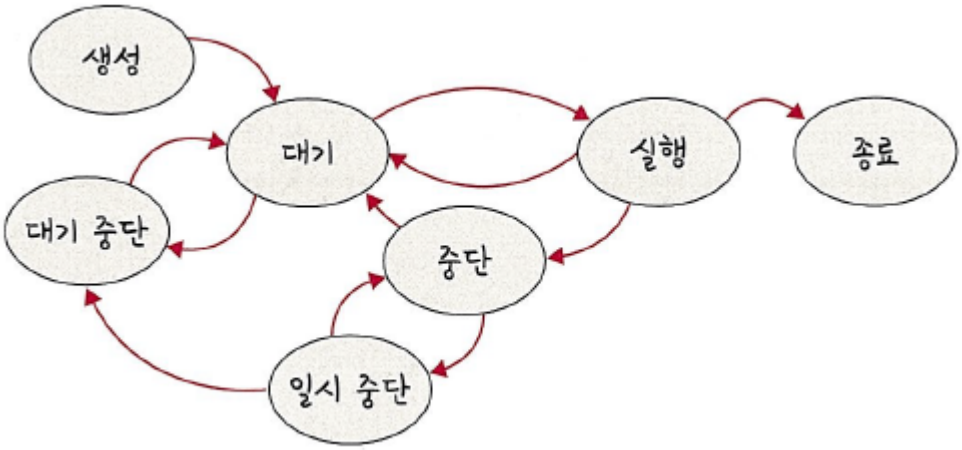
- **동적 라이브러리**

  프로그램 실행 시 필요할 때만 DLL 이라는 함수 정보를 통해 참조하는 방식

  메모리 효율성에서의 장점이 있으나, 외부 의존도가 높아진다는 단점이 있음

프로세스의 상태

프로세스의 상태는 여러가지 상태값을 가짐



생성 상태 (Create)

프로세스가 생성된 상태

fork() 또는 exex() 함수를 통해 생성

이때 PCB 가 할당됨 → 뒤에 나옴

fork()

부모 프로세스의 주소 공간을 그대로 복사하며 새로운 자식 프로세스를 생성하는 함수

주소 공간만 복사함

부모 프로세스의 비동기 작업 등을 상속하지는 않음

exec()

새롭게 프로세스를 생성하는 함수

대기 상태 (Ready)

메모리 공간이 충분하면 메모리를 할당받고 아니면 그대로 대기하고 있으며 CPU 스케줄러로부터 CPU 소유권이 넘어오기를 기다리는 상태

대기 중단 상태 (Ready Suspended)

메모리 부족으로 일시 중단된 상태

실행 상태 (Running)

CPU 소유권과 메모리를 할당받고 인스트럭션을 수행 중인 상태

CPU burst 가 일어났다고도 표현함

중단 상태 (Blocked)

어떤 이벤트가 발생한 이후 기다리며 프로세스가 차단된 상태

IO 디바이스에 의한 인터럽트로 이런 현상이 많이 발생하기도함

ex) 프린트 인쇄 버튼 클릭 시 프로세스가 잠깐 멈춘듯한 경우

일시 중단 상태 (Blocked Suspended)

대기 중단과 유사함

중단된 상태에서 프로세스가 실행되려고 했으나, 메모리 부족으로 일시 중단된 상태

종료 상태 (Terminated)

메모리와 CPU 소유권을 모두 놓고 가는 상태

자연스럽게 종료되는 경우도 있으나, 부모 프로세스가 자식 프로세스를 강제시키는 비자발적 종료 (Abort)로 종료되는 것도 있음

비자발적 종료 (Abort)

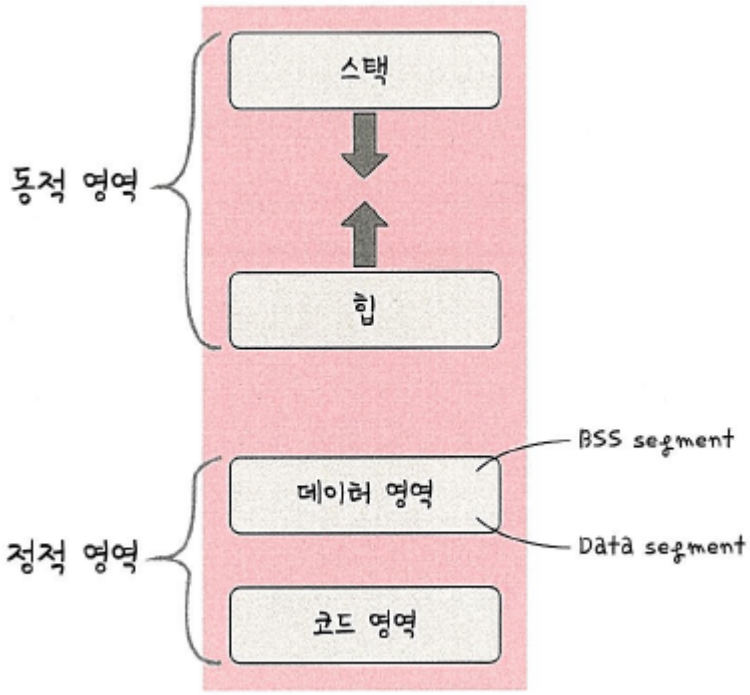
자식 프로세스에 할당된 자원의 한계치를 넘어서는 경우

부모 프로세스가 종료된 경우

사용자가 명령어로 프로세스를 종료할 때 발생

프로세스의 메모리 구조

OS는 프로세스에 적절한 메모리를 할당하는데, 아래 구조를 기반으로 할당함



스택 (Stack)

지역변수, 매개변수, 함수가 저장됨

컴파일 시 크기가 결정되며, 동적인 특징을 가짐

함수가 함수를 재귀적으로 호출하며 동적으로 크기가 늘어날 수 있음

이때, 힙과 스택의 메모리 영역이 겹치면 안되므로 힙과 스택 사이의 공간을 비워 놓음

힙 (Heap)

동적 할당 시 사용되며 런타임 시 크기가 결정됨

동적인 특징을 가짐

데이터 영역

전역변수, 정적변수가 저장되고, 정적인 특징을 갖는 프로그램이 종료되면 사라지는 변수가 들어있는 영역

정적인 특징을 가짐

BSS 영역와 Data 영역으로 나뉨

BSS 영역

초기화가 되지 않은 변수가 0으로 초기화되어 저장됨

Data 영역

0이 아닌 다른 값으로 할당된 변수들이 저장됨

코드 영역

프로그램에 내장되어 있는 소스코드가 들어가는 영역

수정 불가능한 기계어로 저장되어 있으며, 정적인 특징을 가짐

PCB (Process Control Block)

OS 에서 프로세스에 대한 메타데이터를 저장한 데이터. **프로세스 제어 블록.**

프로세스가 생성되면 OS는 해당 PCB를 생성함

프로그램이 실행되면 프로세스가 생성되고, 프로세스 주소 값들에 스택, 힙 등의 구조를 기반으로 메모리가 할당됨

이 프로세스의 메타 데이터들이 PCB에 저장되어 관리됨

이는 프로세스의 중요한 정보를 포함하므로 일반 사용자가 접근할 수 없도록 커널 스택의 가장 앞부분에서 관리됨

PCB의 구조

- **프로세스 스케줄링 상태**

  준비, 일시중단 프로세스가 CPU에 대한 소유권을 얻은 이후 또는 이후 경과된 시간과 같은 기타 스케줄링 정보

- **프로세스 ID**

  프로세스 ID, 해당 프로세스의 자식 프로세스 ID

- **프로세스 권한**

  컴퓨터 자원 또는 IO 디바이스에 대한 권한 정보

- **프로그램 카운터**

  프로세스에서 실행해야할 다음 명령어의 주소에 대한 포인터

- **CPU 레지스터**

  프로세스를 실행하기 위해 저장해야할 레지스터에 대한 정보

- **CPU 스케줄링 정보**

  CPU 스케줄러에 의해 중단된 시간 등에 대한 정보

- **계정 정보**

  프로세스 실행에 사용된 CPU 사용량, 실행한 유저의 정보

- **IO 상태 정보**

  프로세스에 할당된 IO 디바이스 목록

컨텍스트 스위칭 (Context Switching)

PCB를 교환하는 과정

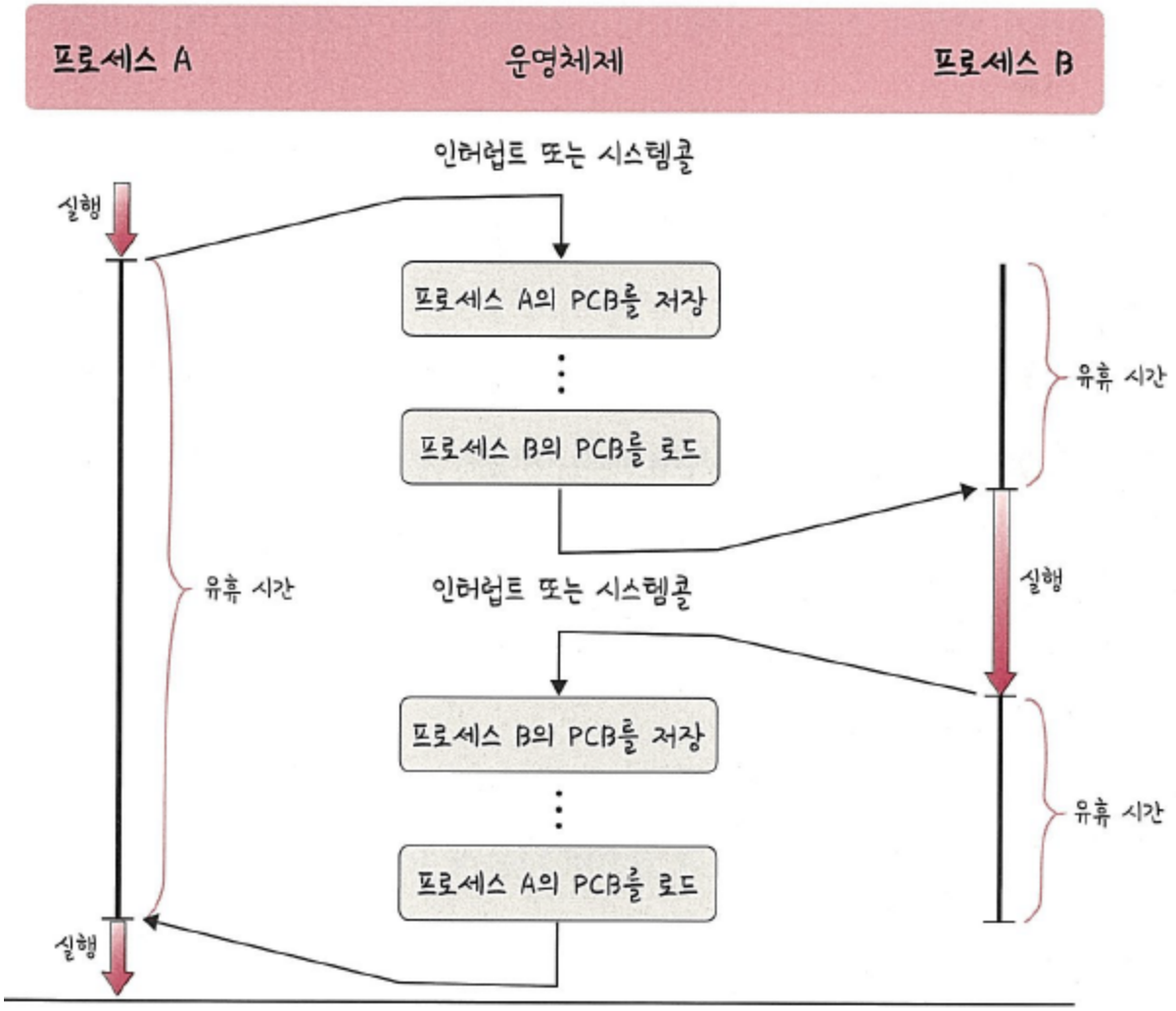
한 프로세스에 할당된 시간이 끝나거나, 인터럽트에 의해 발생

컴퓨터는 많은 프로그램을 동시에 실행하는 것처럼 보이나, 어떠한 시점에서 실행되고 있는 프로세스는 단 한개임

많은 프로세스가 동시에 구동되는 것처럼 보이는 것은 다른 프로세스와의 컨텍스트 스위칭이 아주 빠른 속도로 실행되기 때문임

현대 컴퓨터는 멀티코어의 CPU를 가지므로 한 시점에 한 개의 프로그램이라는 설명은 틀린 설명임

그러나, 컨텍스트 스위칭 설명은 싱글 코어를 기준으로 설명함



위와 같이 한개의 프로세스 A가 실행하다 멈추고, 프로세스 A의 PCB를 저장하고 다시 프로세스 B를 로드하여 실행함

그리고 다시 B의 PCB를 저장하고 A 의 PCB를 로드함

컨텍스트 스위칭이 일어날 때 유휴 시간 (Idle Time)이 발생하는 것을 볼 수 있음

또한, 캐시미스라는 컨텍스트 스위칭에서 드는 비용이 있음

캐시미스

컨텍스트 스위칭이 일어날 때 프로세스가 가지고 있는 메모리 주소가 그대로 있으면 잘못된 주소 변환이 생기므로 캐시 클리어 과정을 겪게됨

이로인해 캐시미스가 발생함

스레드에서의 컨텍스트 스위칭

스레드는 스택 영역을 제외한 모든 메모리를 공유하므로, 스레드 컨텍스트 스위칭의 경우 비용이 더 적고 시간도 더 적게 걸림

멀티프로세싱

멀티프로세스를 통해 동시에 두가지 이상의 일을 수행할 수 있는 것

하나 이상의 일을 병렬로 처리할 수 있음

특정 프로세스의 메모리, 프로세스 중 일부에 문제가 발생하더라도 다른 프로세스를 이용해 처리할 수 있으므로, 신뢰성이 높음

웹 브라우저의 멀티프로세스 구조

- **브라우저 프로세스**

  주소 표시줄, 북마크 막대, 뒤로 가기 버튼, 앞으로 가기 버튼 등을 담당

  네트워크 요청이나 파일 접근 같은 권한을 담당

- **렌더러 프로세스**

  웹 사이트가 보이는 부분의 모든 것을 제어

- **플러그인 프로세스**

  웹 사이트에서 사용하는 플러그인 제어

- **GPU 프로세스**

  GPU를 이용해서 화면을 그리는 부분을 제어

IPC (Inter Process Communication)

프로세스끼리 데이터를 주고받고 공유 데이터를 관리하는 매커니즘

멀티프로세스는 IPC가 가능

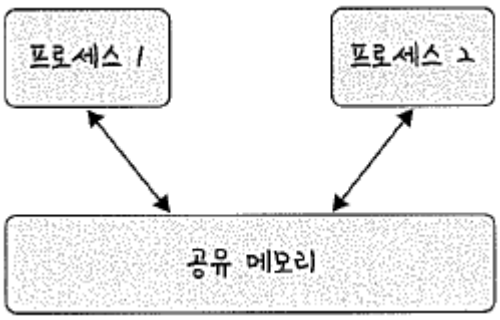
클라이언트와 서버를 예로 들 수 있음

클라이언트는 데이터를 요청하고, 서버는 클라이언트의 요청에 응답하는 것

종류로는 공유 메모리, 파일, 소켓, 익명 파이프, 명명 파이프, 메시지 큐가 있음

메모리가 완전히 공유되는 스레드보다는 속도가 낮음

- **공유 메모리 (Shared Memory)**



  여러 프로세스에 동일한 메모리 블록에 대한 접근 권한이 부여되어 프로세스가 서로 통신할 수 있도록 공유 버퍼를 생성하는 것

  기본적으로는 각 프로세스의 메모리를 다른 프로세스가 접근할 수 없으나, 공유 메모리를 통해 여러 프로세스가 하나의 메모리를 공유 할 수 있음

  IPC 방식 중 어떠한 매개체를 통해 데이터를 주고 받는 것이 아니라 메모리 자체를 공유함

  불필요한 데이터 복사의 오버헤드가 발생하지 않아 가장 빠름

메모리 영역을 여러 프로세스가 공유하므로 동기화가 필요함

하드웨어 관점에서의 공유 메모리는 RAM을 가리키기도 함

- **파일 (File)**

  디스크에 저장된 데이터 또는 파일 서버에서 제공한 데이터

  이를 기반으로 프로세스 간 통신이 이루어짐

- **소켓 (Socket)**

  동일한 컴퓨터의 다른 프로세스나 네트워크의 다른 컴퓨터로 네트워크 인터페이스를 통해 전송하는 데이터

  TCP 와 UDP가 있음

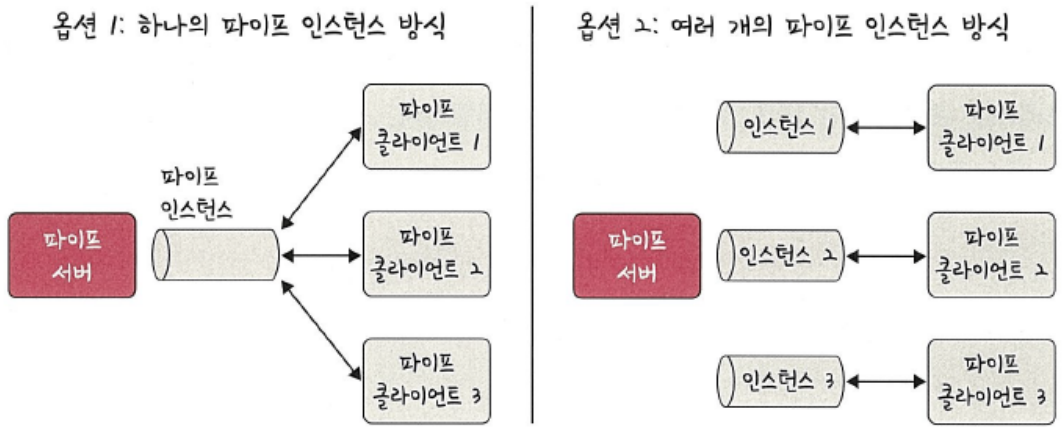
- **익명 파이프 (Unamed Pipe)**

  프로세스 간에 FIFO 방식으로 읽히는 임시공간인 파이프를 기반으로 데이터를 주고 받음

  단방향 방식의 읽기 전용, 쓰기 전용 파이프를 만들어서 동작하는 방식

  부모, 자식 프로세스 간에만 사용 가능. 다른 네트워크에선 사용 불가

- **명명된 파이프 (Named Pipe)**



파이프 서버와 하나 이상의 파이프 클라이언트 간 통신을 위한 명명된 단방향 또는 이중 파이프

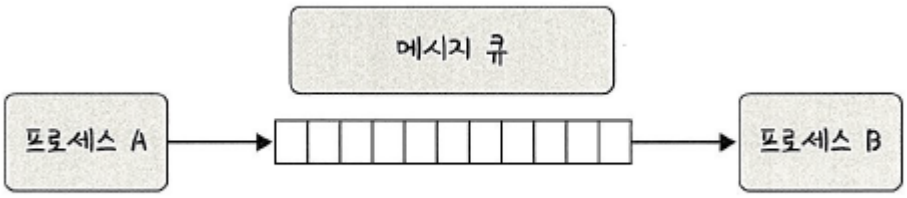
클라이언트 서버 통신을 위한 별도의 파이프를 제공하며, 여러 파이프를 동시에 사용 가능

컴퓨터의 프로세스끼리 또는 다른 네트워크 상의 컴퓨터와도 통신이 가능함

일반적으로 서버용 파이프와 클라이언트용 파이프로 구분해서 작동

하나의 인스턴스를 열거나 여러개의 인스턴스를 기반으로 통신함

- **메시지 큐**



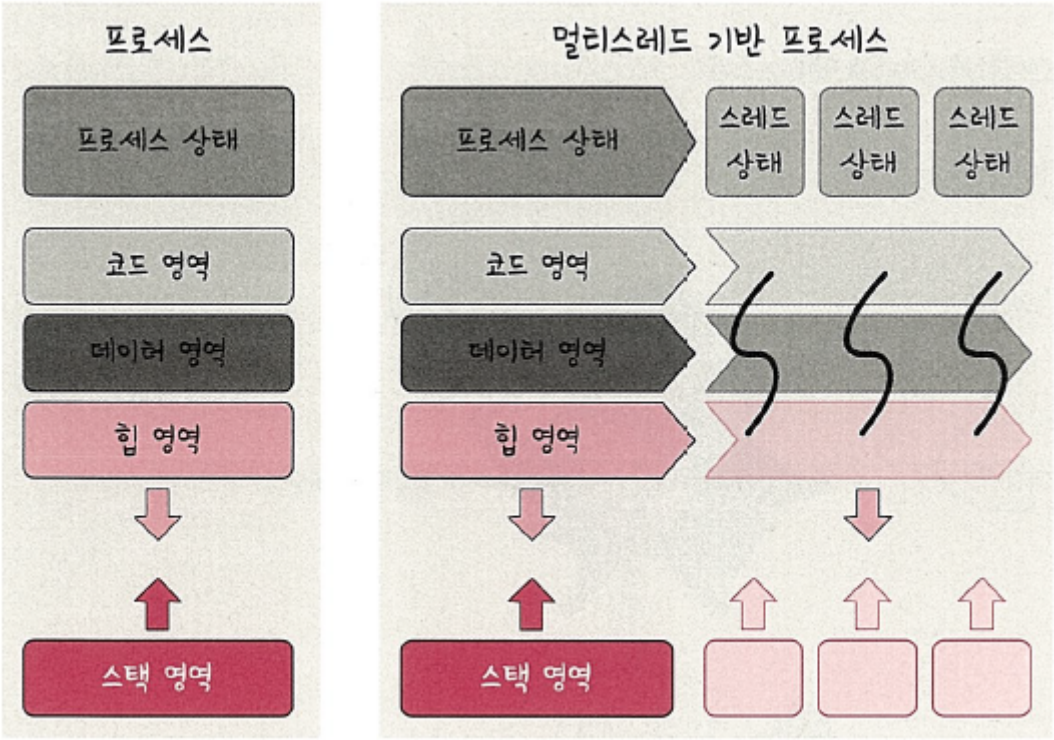
메시지를 큐 데이터 구조 형태로 관리하는 것

커널의 전역변수 형태 등 커널에서 전역적으로 관리되며 다른 IPC 방식에 비해서 사용 방법이 매우 직관적이고 간단함

다른 코드의 수정 없이 단지 몇줄의 코드를 추가시켜 간단하게 메시지 큐에 접근 가능한 장점이 있음

공유 메모리를 통한 IPC 구현 시 쓰기 및 읽기 빈도가 높으면 동기화로 인하여 기능 구현이 매우 복잡해지는데, 이때 대안으로 메시지 큐를 사용하기도 함

스레드와 멀티스레딩



스레드 (Thread)

프로세스의 실행 가능한 가장 작은 단위

프로세스는 여러 스레드를 가질 수 있음

코드, 데이터, 스택, 힙을 각각 생성하는 프로세스와 달리 스레드는 코드, 데이터, 힙은 스레드끼리 서로 공유함

그외의 영역은 각각 생성됨

멀티스레딩

프로세스 내 작업을 멀티 스레드로 처리하는 기법

스레드끼리 서로 자원을 공유하므로 효율성이 높음

ex) 웹 요청 처리 시

> 새 프로세스를 생성하는 대신, 스레드를 사용하는 웹 서버의 경우 훨씬 적은 리소스를 소비

한 스레드가 중단되어도, 다른 스레드는 실행 상태일 수 있으므로 중단 없는 빠른 처리가 가능

동시성에도 장점이 있음

- 동시성: 서로 독립적인 작업들을 작은 단위로 나누고 동시에 실행되는 것처럼 보여주는 것

그러나, 한 스레드에 문제가 생기면 다른 스레드에도 영향을 끼쳐, 스레드로 이루어진 프로세스에 영향을 줄 수 있는 단점이 있음

ex) 웹 브라우저의 렌더러 프로세스

이 프로세스 내에는 메인 스레드, 워커 스레드, 컴포지터 스레드, 레스터 스레드가 존재함

공유 자원과 임계 영역

공유 자원 (Shared Resource)

시스템 안에서 각 프로세스, 스레드가 함께 접근 가능한 모니터, 프린터, 메모리, 파일, 데이터 등의 자원이나 변수 등

경쟁 상태 (Race Condition)

공유 자원을 두 개 이상의 프로세스가 동시에 읽거나 쓰는 상황

동시에 접근을 시도할 때 접근의 타이밍이나 순서 등이 결괏값에 영향을 줄 수 있는 상태

임계 영역 (Critical Section)

공유 자원 접근 시 순서 등의 이유로 결과가 달라지는 영역

해결방안

뮤텍스, 세마포어, 모니터 세가지가 있음

위 세가지 모두 상호 배제, 한정 대기, 융통성 이라는 조건을 만족

- **상호배제**: 한 프로세스가 임계 영역에 들어갔을 때 다른 프로세스는 들어갈 수 없음
- **한정대기**: 특정 프로세스가 영원히 임계 영역에 들어가지 못하면 안됨
- **융통성**: 한 프로세스가 다른 프로세스의 일을 방해해서는 안됨

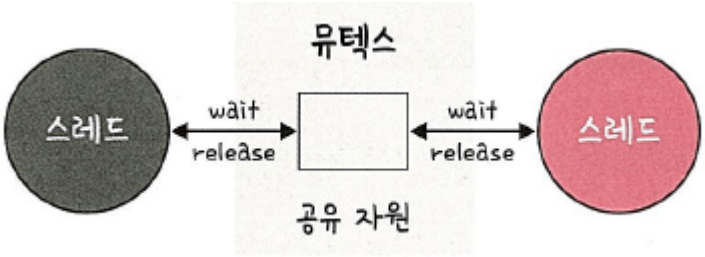
이 방법에 토대가 되는 매커니즘은 **잠금(lock)**

ex) 화장실을 임계구역이라고 가정

화장실에 A라는 사람이 들어간 다음 문을 잠금

다음 사람이 이를 기다리다가 나오면 사용이 가능함

뮤텍스 (Mutex)

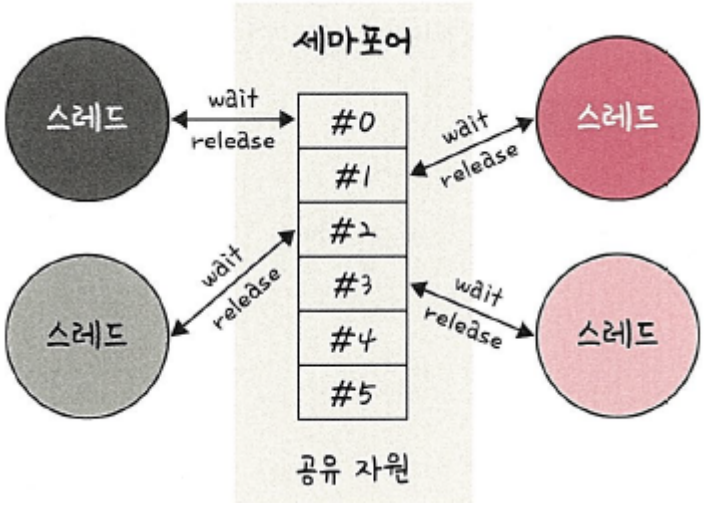


공유 자원을 사용하기 전에 설정하고 사용한 후에 해제하는 잠금

잠금이 설정되면 다른 스레드는 잠긴 코드 영역에 접근 불가능

하나의 상태 (잠금 또는 잠금 해제) 만을 가짐

세마포어 (Semaphore)



일반화된 뮤텍스

간단한 정수 값과 두가지 함수 wait (P 함수) 및 signal(V 함수)로 공유자원에 대한 접근을 처리

- `wait()` : 자신의 차례가 올 때까지 기다리는 함수
- `signal()` : 다음 프로세스로 순서를 넘겨주는 함수

프로세스가 공유 자원에 접근하면 세마포어에서 `wait()` 작업을 수행하고, 프로세스가 공유 자원을 해제하면 세마포어에서 `signal()` 작업을 수행

세마포어에는 조건 변수가 없고, 프로세스가 세마포어 값을 수정할 때 다른 프로세스는 동시에 세마포어 값을 수정할 수 없음

세마포어의 종류

- **바이너리 세마포어**

  0과 1의 두 가지 값만 가질 수 있는 세마포어

  뮤텍스와 구현의 유사성을 가짐

  그러나, 뮤텍스는 리소스에 대한 접근 동기화에 사용되는 잠금 매커니즘

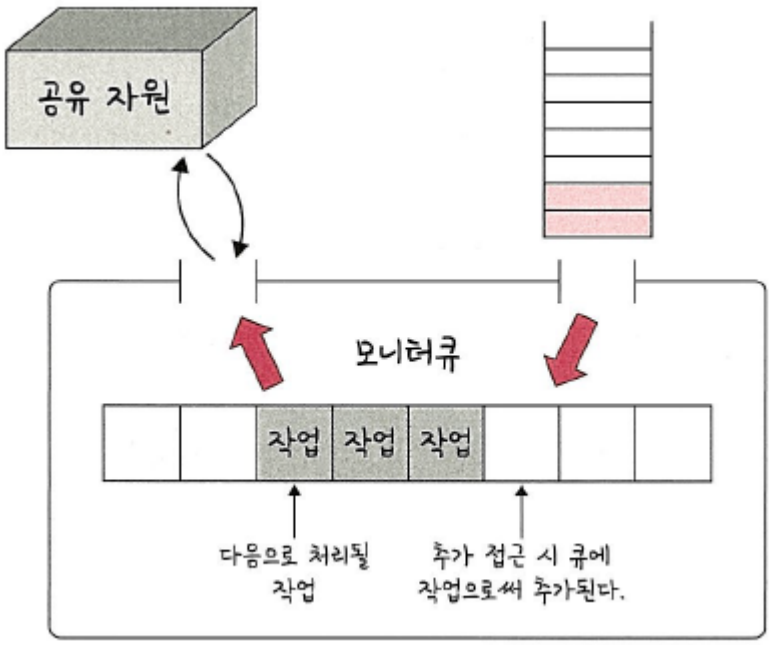
  세마포어는 신호를 기반으로 상호배제가 일어나는 신호 매커니즘

- **카운팅 세마포어**

  여러 개의 값을 가질 수 있는 세마포어

  여러 자원에 대한 접근 제 어시 사용됨

모니터



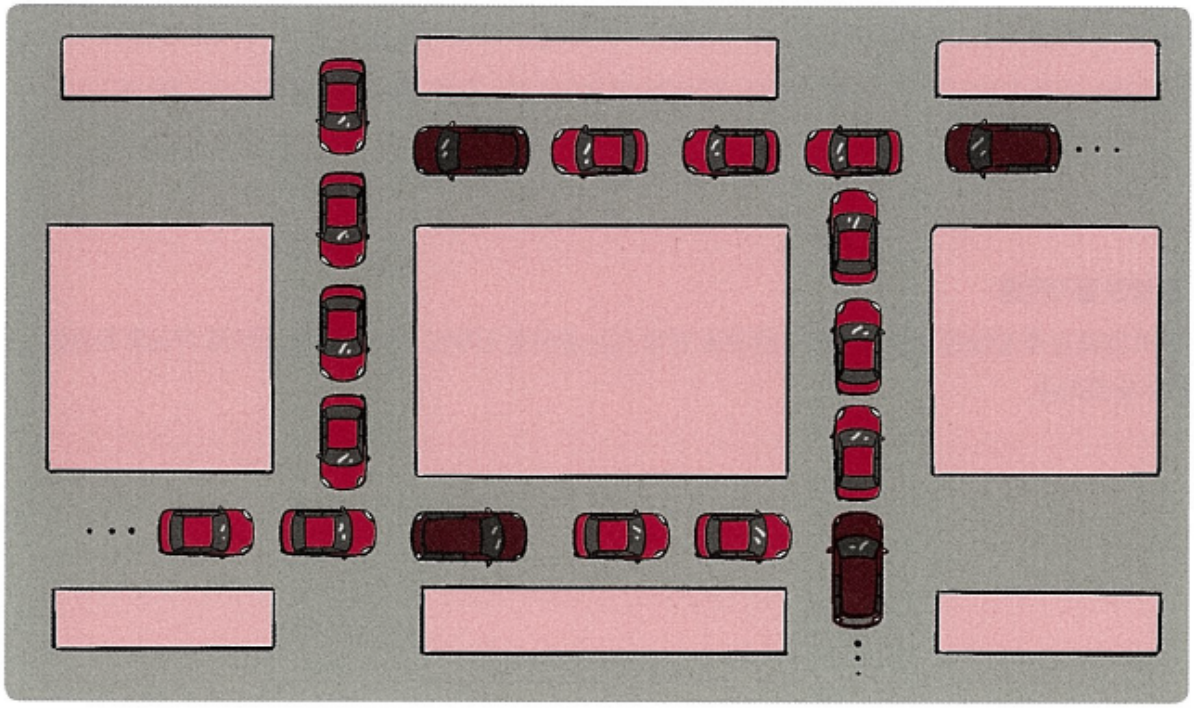
둘 이상의 스레드나 프로세스가 공유 자원에 안전하게 접근 가능하도록 공유 자원을 숨기고 해당 접근에 대해 인터페이스만 제공함

모니터는 모니터 큐를 통해 공유자원에 대한 작업들을 순차적으로 처리함

세마포어보다 구현하기 쉬우며, 상호 배제가 자동으로 이루어짐

세마포어에서는 상호 배제를 명시적으로 구현해야함

교착 상태 (Deadlock)



두 개 이상의 프로세스들이 서로가 가진 자원을 기다리며 중단된 상태

원인

- **상호 배제** : 한 프로세스가 자원을 독점하고 있으며, 다른 프로세스들은 접근 불가능
- **점유 대기** : 특정 프로세스가 점유한 자원을 다른 프로세스가 요청하는 상태
- **비선점** : 다른 프로세스의 자원을 강제적으로 가져올 수 없음
- **환형 대기** : 프로세스 A는 B의 자원을 요구하고, B는 A의 자원을 요구하는 등 서로가 서로의 자원을 요구하는 상황

해결 방법

1. 자원 할당 시 애초에 조건이 성립되지 않도록 설계
2. 교착 상태 가능성이 없을 때만 자원이 할당되며, 프로세스당 요청할 자원들의 최대치를 통해 자원 할당 가능 여부를 파악하는 `은행원 알고리즘`을 사용
    - 은행원 알고리즘

      총 자원의 양과 현재 할당한 자원의 양을 기준으로 안정 또는 불안정 상태로 나누고 안정 상태로 가도록 자원을 할당하는 알고리즘
3. 교착 상태 발생 시 사이클이 있는지 찾아보고 이에 관련된 프로세스를 한개씩 지움
4. 교착 상태는 매우 드물게 일어나므로, 이를 처리하는 비용이 더 커서 교착 상태가 발생하면 사용자가 작업을 종료함. 현대 OS 에서 채택한 방식

   ex) `응답없음`