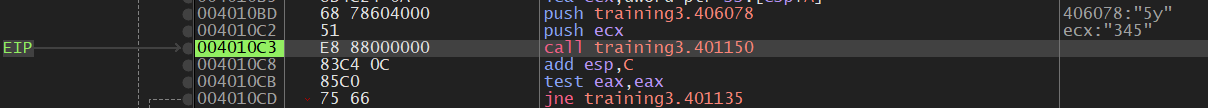
**실습 3**

텍스트, 모니터, 실내, 화면이(가) 표시된 사진

자동 생성된 설명

12345를 입력했을 때 2와 ‘a’를 비교함 → 정답의 두번째 글자가 a 라는 것을 확인

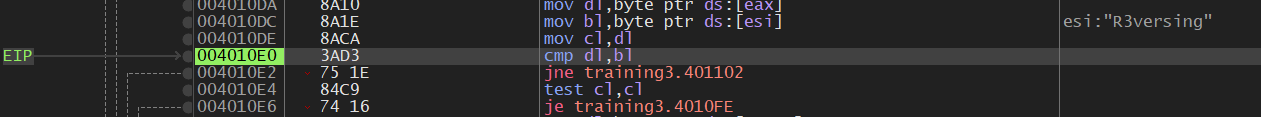


텍스트, 모니터, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

3과 “5y”를 함수를 통해서 비교, / test eax, eax를 통해 eax가 0이여야 함을 확인

→ 정답의 세번째 글자부터 5y 라는 것을 확인





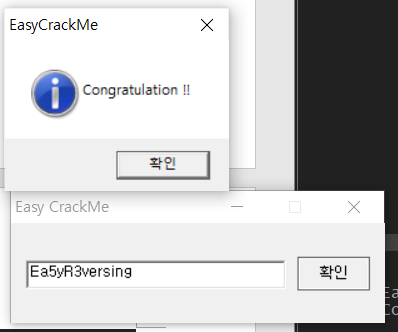
5와 “R3versing”을 비교함 → 정답의 다섯번째 글자부터 R3versing 이라는 것을 확인

텍스트, 스크린샷, 실내, 모니터이(가) 표시된 사진

자동 생성된 설명

1과 ‘E’를 비교함 → 정답의 첫번째 글자가 E 이라는 것을 확인

정답 : Ea5yR3versing



**Jrev1 풀이**

텍스트, 실내, 점수판이(가) 표시된 사진

자동 생성된 설명

Fopen 함수의 인자로 12114, 12110 을 넣어줌

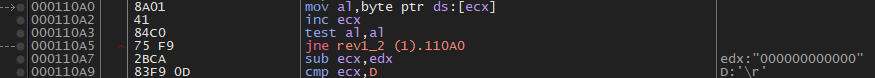


파일 경로와 “r” 이 들어감을 확인

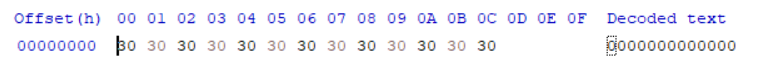
텍스트이(가) 표시된 사진

자동 생성된 설명

해당 경로에 temp 라는 이름의 파일 생성



Cmp D (=14)와 비교해주기에 파일 내부의 글자수를 14개로 생성함

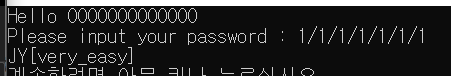




텍스트이(가) 표시된 사진

자동 생성된 설명

예시로 123을 입력했을 때 cmp dl, byte ptr ds:[ecx]를 통해 “1/1/1/1/1/1/1” 과 비교한다는 것을 확인



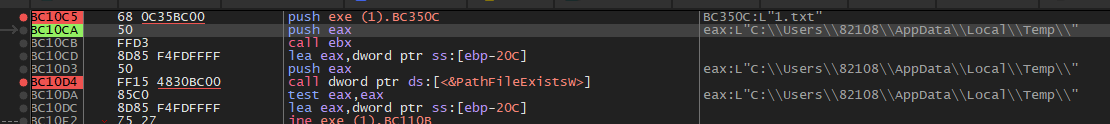
1/1/1/1/1/1/1 을 입력하면 정답

**Jrev 2 풀이**

텍스트이(가) 표시된 사진

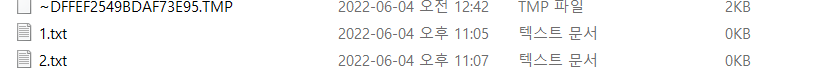
자동 생성된 설명

1.txt 랑 2.txt를 확인하고 브레이크를 걸었음



eax:L"C:\\Users\\82108\\AppData\\Local\\Temp\\"

를 확인함



해당 경로에서 실행시킬 때마다 1.txt 랑 2.txt 가 생성되는 것을 확인

82108 → root 로 바꾸면 정답

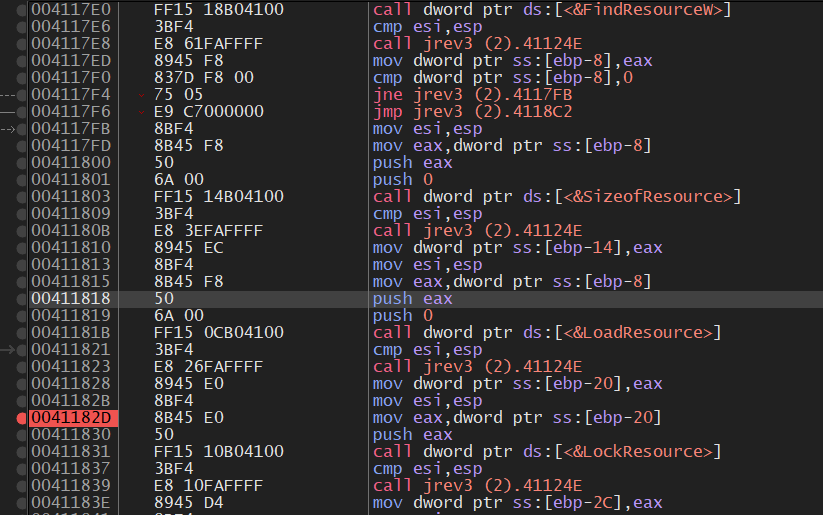
**Jrev 3 풀이**

텍스트이(가) 표시된 사진

자동 생성된 설명

Call LoadResource 로 LoadResource의 반환값에 flag가 존재한다는 것을 확인

→ 그 앞뒤의 call하는 함수들을 확인해서 어떤 의미인지 확인해보았음



FindResource 함수로 분리해 내려고 하는 리소스를 찾음

SizeofResource 함수로 리소스의 크기를 구함

LoadResource로 리소스를 로드함

LockResource로 LoadResource한 메모리의 실제 포인터를 반환한다는 것을 확인

→ 실제 포인터가 리턴됨.

**Jrev 4 풀이**

텍스트, 스크린샷, 점수판이(가) 표시된 사진

자동 생성된 설명

00411933 에서 예외가 발생했기에 앞뒤의 코드들을 살펴봄

텍스트이(가) 표시된 사진

자동 생성된 설명

VirtualQuery와 VirtualProtect 라는 함수를 확인

VirtualProtect 위의 push로 인자들을 넘겨줌을 확인

테이블이(가) 표시된 사진

자동 생성된 설명

메모리 시작주소 : FFFFFFFF

메모리 크기 : 1000

메모리 보호 상수 : 2

변경전 상태를 저장할 변수 포인터 : 19FED0 임을 확인

메모리 보호 상수를 2 → 0x80으로 변경함으로써 권함을 줌

실행 후 플래그 찾음!

**#1 커널영역과 유저영역 차이**

일반 프로그램을 실행시키기 위해 필요한 메모리 공간과 운영체제의 실행을 위한 메모리 공간을 분리 X 하면 관리하기 힘들어짐

→ 유저 영역과 커널 영역을 분리하는 이유

**유저 영역** : 프로그램이 동작하기 위해 사용되는 메모리 공간

→ 스택 영역, 힙 영역, 데이터 영역, 코드 영역으로 나뉨

사용자 모드 : 사용자 영역의 응용 프로그램이 사용하는 모드

사용자 영역의 응용프로그램이 커널 영역의 기능을 사용하기 위해서는 시스템 콜이 필요함

유저 어플리케이션은 사용자 모드에서만 실행 o

+ **시스템 콜**이란? : 프로세스가 하드웨어에 직접 접근해서 필요한 기능을 사용할 수 있게함

Ex) 쉘에서 cat이라는 명령어를 입력, cat을 살펴보면 안에는 파일을 open하거나 read 해주는 시스템 콜이 있음

**커널 영역** : 메모리에서 유저 영역을 제외한 영역

커널이 위치함

커널 모드 : 운영체제가 CPU 제어권을 가지고 운영체제 코드를 실행하는 모드

운영체제 코드는 커널 모드에서만 실행 o ,

z커널이 실행될 때는 커널모드에 진입되어 메모리 공간 어디든 접근할 수 있음

→ 유저 영역과 커널 영역에서 전부 접근 가능

+ **커널**이란? : 운영체제의 핵심부로 컴퓨터 자원(CPU, 메모리, 파일 등)들을 관리함

프로그램이 동작할 때 필요한 부분은 메모리에 올라가서 사용하게 됨

→ 메모리에 상주하는 운영체제의 부분 = 커널

커널은 컴퓨터 자원을 관리하며 사용자와의 상호작용을 하지 않음

→ 쉘을 통해서 커널에 명령을 내림

**#2 가상 메모리와 물리 메모리**

**가상 메모리**

방식 : 각 프로그램에 실제 메모리 주소가 아닌 가상의 메모리 주소를 주는 방식

실제로 프로그램 수행에 필요한 부분만 메모리에 올려놓음

→ 주기억장치의 용량보다 큰 프로그램 파일도 사용자가 메모리에 올려놓을 수 있음 (주기억장치의 크기보다 더 큰 공간 사용 가능)

메모리 관리 기능을 통해 물리 주소로 변환되어 실제 물리 메모리에 매핑됨

프로세스간 공간 불리로 프로세스 이슈가 전체 시스템에 영향 X

+ 메모리 매핑이란?

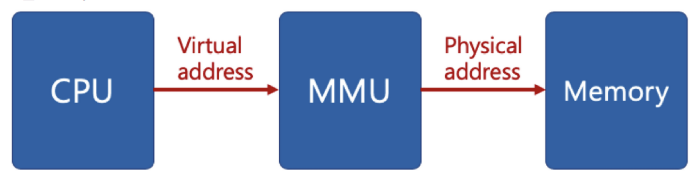
[파일](https://www.webopedia.com/definitions/file/), [코드](https://www.webopedia.com/definitions/code/) 또는 [개체를](https://www.webopedia.com/definitions/object/) 시스템 [메모리](https://www.webopedia.com/definitions/memory/)로 가져오도록 요청하는 컴퓨터 [프로그래밍](https://www.webopedia.com/definitions/program/)의 프로세스 또는 명령

가상 메모리 시스템을 사용하기 위해선 가상 주소 / 물리 주소가 필요

가상 주소 : 프로세스가 참조하는 주소

물리 주소 : 실제 메모리 주소

MMU : 가장 주소 메모리 접근이 필요할 때, 해당 주소를 물리 주소 값으로 변환해주는 하드웨어 장치



**물리 메모리**

접근 방식) CPU가 논리 주소를 읽음, 논리 주소를 통해 물리 주소를 읽고 물리 메모리에 접근

(논리 메모리의 순서와 크기를 따름)

CPU는 물리 메모리에 적재된 코드만 실행 가능

논리 주소와 물리 주소간 주소 변환 과정의 주체 = MMU

<분할 방식>

1. **고정 분할 방식**

물리적 메모리를 몇 개의 부분으로 분할함 (분할의 크기는 동일할 수도 있고 아닐 수도 있음)

분할 당 하나의 프로그램을 적재시킴

→ 단편화 발생

1. **가변 분할 방식**

프로그램의 크기대로 메모리를 분할해서 할당함 (분할의 크기, 개수가 동적으로 변함)

프로세스를 순서대로 메모리에 쌓음

→ 프로그램이 실행되고 종료되는 과정에서 외부 단편화가 발생 할 수도 있음

외부 단편화 : 프로그램 크기보다 분할의 크기가 작아 현재 대기중인 프로그램이 올라갈 수 없는 작은 분할

내부 단편화 : 프로그램 크기보다 분할의 크기가 커서 하나의 프로그램이 올라간 분할에 사용되지 않는 공간이 발생

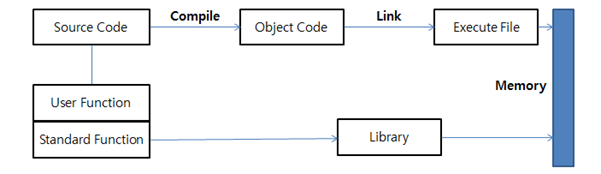
**#3 DLL 이란**

**DLL (Dynamic Link Library, 동적 라이브러리)**

다른 프로그램에서 특정 작업을 수행하기 위해 호출할 수 있는 지침이 들어있는 파일 형식

필요한 함수가 포함된 객체 파일을 꺼내서 실행 파일에 결합

(함수는 표준 라이브러리 파일 안에 어셈블리 코드 형태로 담겨있음)



→ 프로그램의 소스 코드를 컴파일 함

‘동적 링크’ 방식 사용

: 실행 파일에 결합되는 것이 아니라 프로그램 실행 시에 DLL 도 함께 메모리의 공간으로 읽어와 호출될 주소 등을 적절하게 바꾸는 것을 의미

여러 프로그램이 단일 파일에 프로그래밍 된 기능을 공유 가능 & 동시에 수행 가능

→ 사용되는 디스크공간을 줄일 수 있음

DLL 파일은 EXE 파일과 다르게 직접 실행 될 수는 없고 실행중인 다른 코드에 의해 호출되어야 함

프로그램을 여러 개별 구성 요소로 모듈화 할 수 있음

→ 모듈식 아키텍쳐 활용

(+ 모듈식 프로그램 :  대규모 소프트웨어 **프로그램**의 구성 요소를 관리 가능한 부분으로 분리하여 소프트웨어를 설계)

**#4 멀티바이트 유니코드 차이점**

**멀티바이트 :** 하나 이상의 바이트로 표현되는 방식

EX. 영어 1byte, 한글 2byte

멀티바이트 문자 집합은 특정 문자 집합마다 코드 페이지가 존재함

(+ 코드 페이지 : 특정 문자 세트에서 사용 가능한 각 문자를 설명하는 키)

→ 각 나라 언어 페이지로 해석하면 서로 다른 언어가 나와 문자가 깨질 수도 있음

→ 유니코드는 아스키 문자 등 모든 문자들을 한 문자당 2 byte 로 할당함

**유니코드 :** 항상 2바이트 크기로 문자를 표현하는 방식

각 특정 문자는 고유의 유니코드 값을 지님

<코딩 시 유의점>

|  |  |
| --- | --- |
| 멀티바이트 | 유니코드 |
| char | TECHAR |
| strcat\_s() | \_tcscat\_s() |
| strcpy\_s() | \_tcscpy\_s(), \_tcsncpy\_s() |
| strlen() | \_tcslen() |
| sprint\_s() | \_stprintf\_s() |
| “” | TEXT(“ ”) |

+ 아스키 코드 : 모든 문자가 1byte를 차지하며 영어를 표현함

**#5 PE에서 일반적인 섹션 4가지에 대해 조사해오고, 각 섹션마다의 권한(상세)**

**PE (Portable Excutable)**

: Windows 3.1 윈도우 상에서 지원되는 실행파일의 구조

<종류>

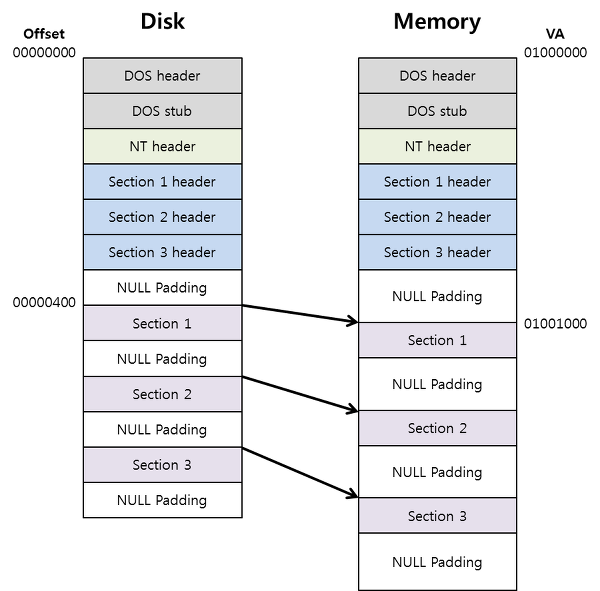
실행 계열, 라이브러리 계열, 드라이버 계열, 오브젝트 파일 계열

**<기본 구조>**

PE 파일 = 헤더 + 바디

헤더: 파일이 실행되기 위해 필요한 정보 저장

바디: 명령어(코드), 데이터, 리소스 등이 저장



섹션의 헤더 : 각 섹션의 파일/메모리에서의 크기, 위치, 속성 등이 정의됨

PE 헤더의 끝부분, 각 섹션의 끝 : Null padding 영역 존재 (파일, 메모리 등을 처리할 때 효율을 높이기 위한 기본 단위 개념으로 사용)

1. ‘text’ 섹션 (Code : 실행, 읽기 권한)

파일을 열었을 때 실행될 코드를 저장함

2. ‘data’ (Data: 비실행, 읽기, 쓰기 권한)

초기화된 전역 변수, static 변수 저장

3. ‘rdata’ (Data: 비실행, 읽기, 쓰기 권한)

Const 변수, 문자열 상수 저장

4 ‘rsrc’ (Resource : 비실행, 읽기 권한)

리소스 저장

**#6 VA, RVA, RAW 조사**

**VA :** 프로세스 가상 메모리의 절대주소

VA = RVA + ImageBase

(+ ImageBase : 가상 메모리 상에서 PE 파일이 로드되는 시작 주소)

가상의 메모리이기에 파일의 시작 주소를 알아야 파일에 접근할 수 있음

→ 실제 주소와 가상 주소를 연결해주는 것이 매핑 / 매핑 되는 위치가 ImageBase

**RVA:** Image Base로부터의 상대 주소

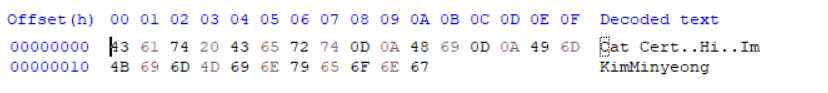
RVA = RAW + VA – PointToRawData = VA – ImageBase

(+ PointToRawData : 섹션의 첫번째 페이지를 가리키는 파일 포인터)

ImageBase는 메모리에서 PE파일이 어디에 로딩 되는지 메모리에 로딩되는 시작 주소를 알려줌

**RAW:** 디스크 상의 파일에서의 주소(오프셋)

RAW = RVA – VA + PointToRawData



→ 맨 왼쪽의 offset 값이 File Offset = RVA

**#7 하드웨어 , 소프트웨어 브레이크포인트 차이**

브레이크 포인트 : 디버깅을 목적으로 실행중인 프로세스를 의도적으로 멈추게 하는 장소

**소프트웨어 브레이크 포인트 :**

디버깅 진행시 흔하게 사용되는 브레이크 포인트 / 한 바이트의 명령으로 프로세스 실행을 중지시킴

**<과정>**

Opcode : 명령코드, 명령어에서 실제로 어떤 동작을 하는지 나타내는 부분

CC : INT3 라는 CPU의 256개의 트랩 중 하나를 발생시키는 명령어

INT : CPU가 예외처리를 하기 위해 인터럽트(예기치 않은 일이 발생해도 중단 X, 계속 업무 처리할 수 있게 하는 기능)를 발생시키는 것

브레이크 포인트 설정 ▷ 해당 명령의 Opcode를 브레이크 포인트 리스트에 저장 ▷ 해당 Opcode를 cc로 변경해서 인터럽트 발생시킴 ▷ cc를 통해 브레이크 포인트 걸린 후 진행 ▷ 브레이크 포인트 리스트에서 저장해 놓은 명령으로 Opcode를 다시 바꿈

**하드웨어 브레이크 포인트 :**

디버그 레지스터(DR)을 사용함

코드 변경이 안 되거나 브레이크 포인트 사용이 적을 때 사용하기에 적합함

|  |  |  |
| --- | --- | --- |
|  | 소프트웨어 | 하드웨어 |
| 코드 | 실행 바이너리의 byte 변경  → CRC 값이 변경 | 디버깅 할 소프트웨어 코드 변경 X |
| 개수 | 브레이크 포인트 횟수 제한 X | 최대 4개까지의 브레이크 포인트 |
|  | INT3 발생 | INT1 발생 |

+ CRC : 순환 중복 검사 (데이터 변경되었는지 확인)

**#8 세마포어 뮤텍스 차이 (상세)**

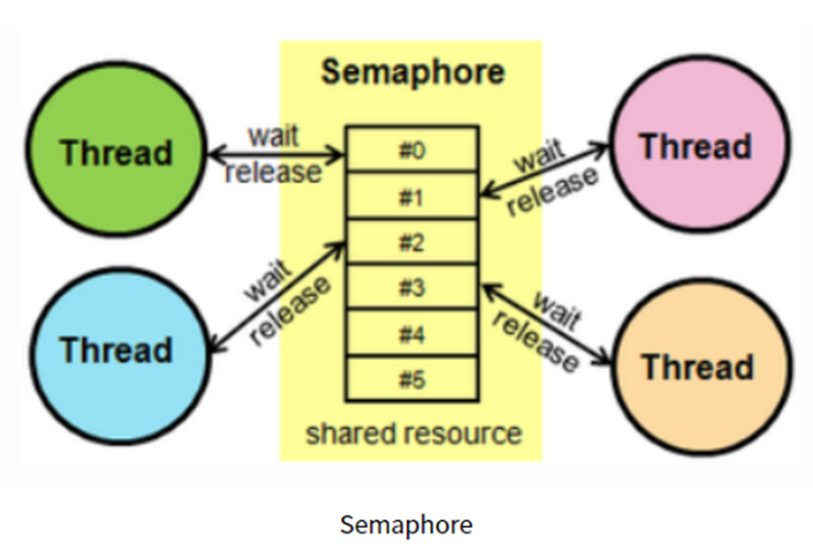
공유된 자원에 여러 프로세스가 동시에 접근 → 문제 발생

→ 데이터를 한번에 하나의 프로세스만 접근하게 하는 동기화 방식

+ 임계 구역 : 프로그램 코드 상에서 공유 자원에 접근하는 부분

대표적인 동기화 도구 : 뮤텍스 / 세마포어

**세마포어** : 공유 자원에 여러 프로세스가 접근하는 것을 제한하는 방법



동기화 대상이 1개 이상일 때 사용

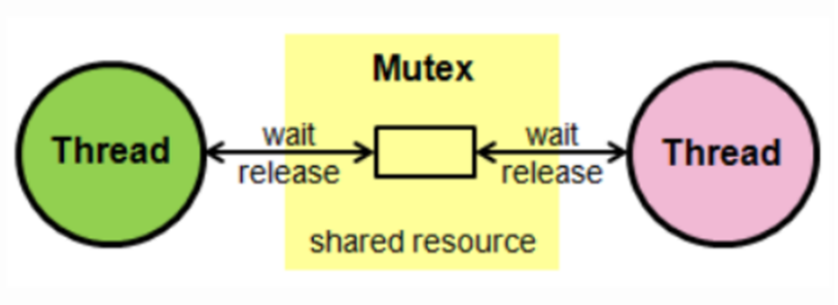
현재 공유 자원의 상태를 나타내는 하나의 값을 두어 상호배제를 달성하는 기법

▷ 정수 값은 접근할 수 있는 최대 허용치만큼 동시에 사용자 접근을 할 수 있게함

현재 수행중인 프로세스가 아닌 다른 프로세스가 세마포어를 해제 할 수 있음

+ 상호배제 : 동시 프로그래밍에서 공유 불가능한 자원의 동시 사용을 피하기 위한 알고리즘

**뮤텍스** : 동시 프로그래밍에서 공유 불가능한 자원의 동시 사용을 피하기 위해 사용하는 알고리즘



동기화 대상이 오직 1개

임계 구역을 가진 스레드들의 실행기간이 서로 겹치지 X / 각각 단독으로 실행되도록 함

하나의 쓰레드만이 들어옴

초기값을 1과 0을 가짐 ▷ 임계영역에 들어갈 때 lock을 걸어서 다른 접근 막음 ▷ 임계영역에 나올 때 lock을 해제

**#9 프로세스와 쓰레드 차이점**

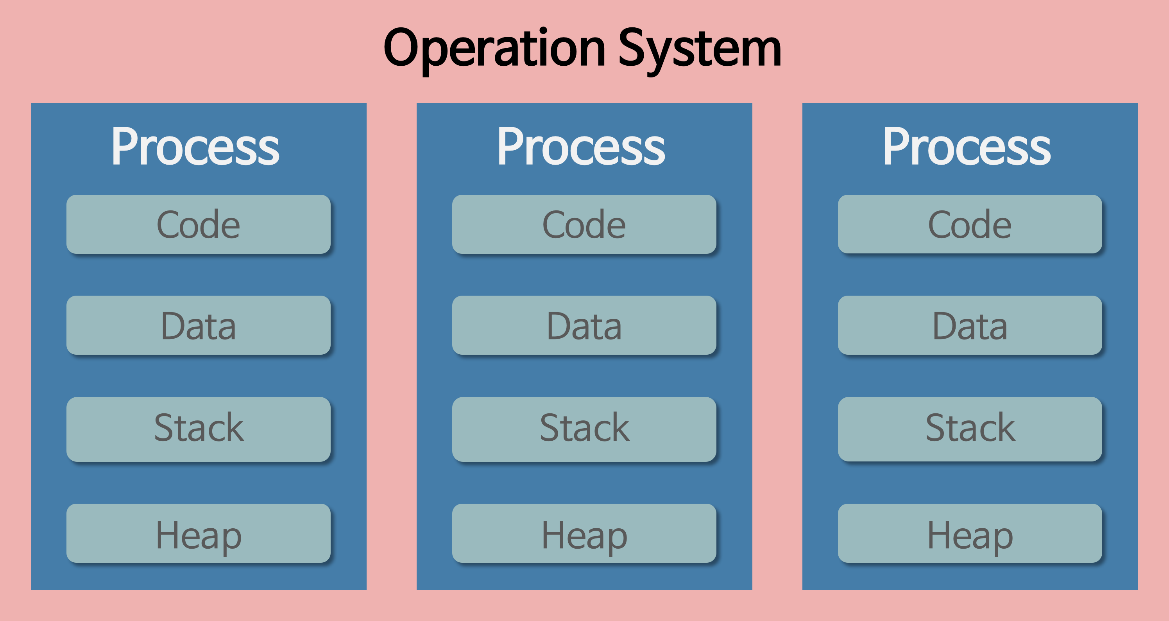
프로그램 ▶ 프로세스 ▶ 스레드

**프로세스** : 메모리에 올라와 실행되고 있는 프로그램 각각의 객체 (작업의 단위)

실행 시 운영체제로부터 프로세서, 메모리 영역(Code, Data, Stack, Heap) , CPU 시간 등의 자원을 할당 받음

같은 프로세스 내의 자원들을 스레드끼리 공유하면서 실행

다른 프로세스의 자원에 접근하기 위해선 프로세스 간의 통신 필요 (EX. 파일)

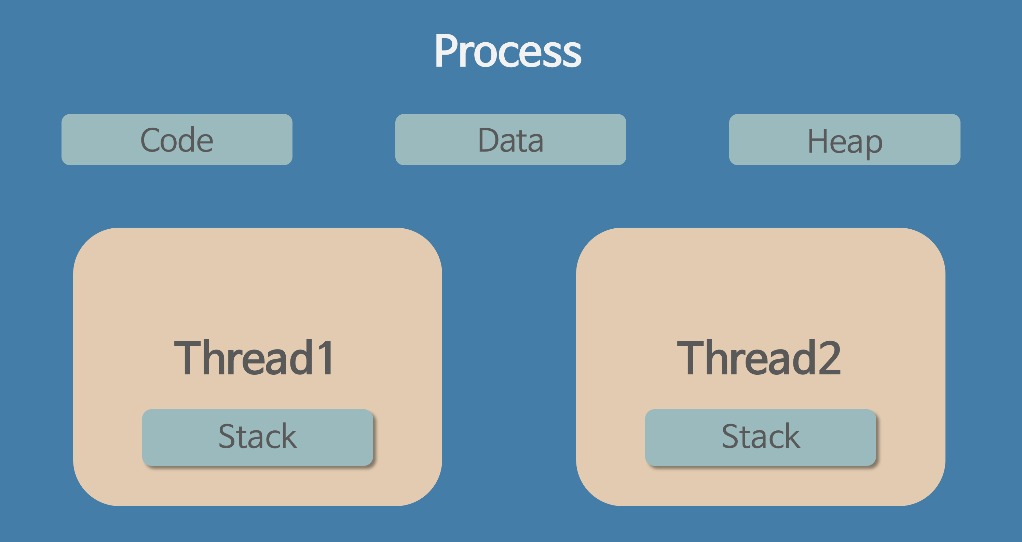


**스레드 :** 프로세스가 할당받은 자원을 이용하는 실행의 단위

프로세스 내에서 stack 부분만 따로 할당 받음, code, data, heap 영역은 공유

(X 프로세스는 다른 프로세스의 메모리에 접근 X)

하나의 프로세스가 생성되면 하나의 (메인) 스레드가 생성됨



사용 이유)

# 작업들 간의 통신 비용 절감

→프로세스 간의 통신 비용 > 하나의 프로세스 내의 여러 스레드 간의 통신 비용

#메모리 절약

→ 1MB 이내의 메모리만 점유

+ 멀티 프로세스와 멀티 스레드

멀티 프로세스 : 하나의 응용 프로그램을 여러 개의 프로세스로 구성 / 각 프로세스는 하나의 작업을 처리함

멀티 스레드 : 하나의 응용 프로그램을 여러 개의 스레드로 구성 / 각 스레드는 하나의 작업을 처리함