Computer Science Study

Operating System

Subject 0. Interrupt
Subject 1. Program
Subject 2. Process
Subject 3. IPC
Subject 4. Thread
Subject 5. Java Thread

홍승택
2024. 09. 23. 19시

made by Seungtaek

subject 0.

More - interrupt

Interrupt handling philosophy in Kernel

- **Do as little as possible in the interrupt handler**
- **Defer non-critical actions till later**
- **Structure: top and bottom halves**
  - Top-half (Hardware Interrupt): do minimum work and return
    - ISR(Interrupt Service Routine)
  - Bottom-half (Software Interrupt): deferred processing
    - softirqs, tasklets, workqueues

Top Half / Bottom Half

Top Half（hardware interrupt)

- it preempts other tasks
- 인터럽트 핸들러에서 최소한의 연산을 수행
- IRQ is typically masked for duration of top half

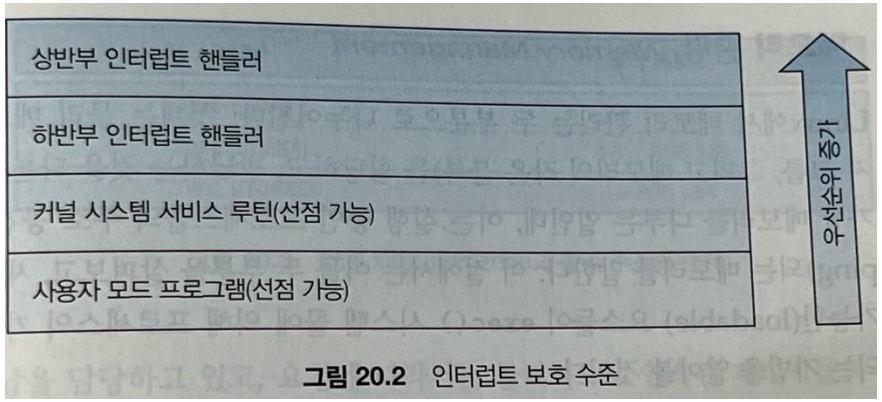
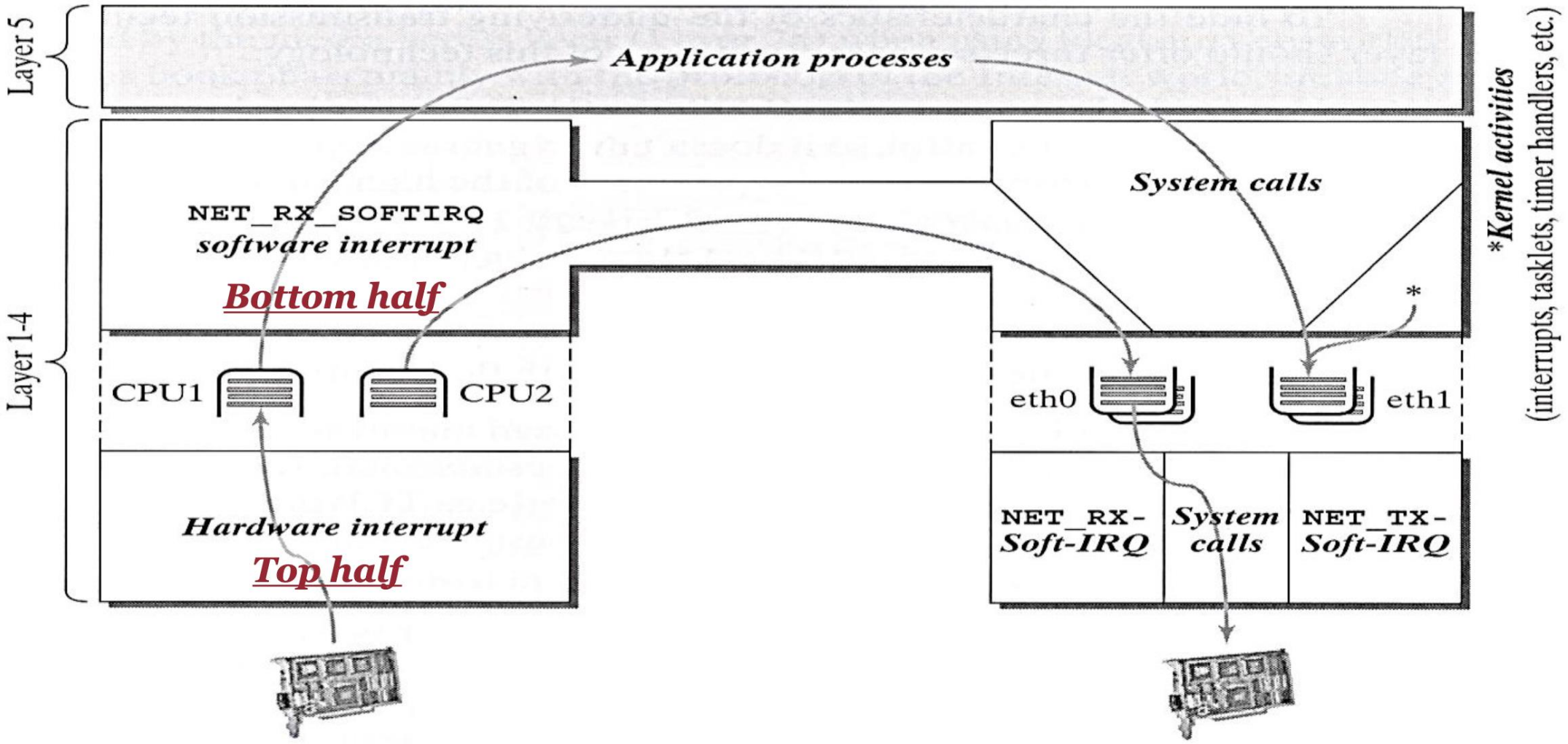


그림 20.2 인터럽트 보호 수준

Bottom Half (software interrupt)

- 인터럽트 처리를 프로세스 레벨에서 수행하는 방식
- Top half 에서는 최소한의 일을 수행하고, 나머지는 Bottom Half 부분에서 처리
- Network transmission and reception is handled with softirq

Network interrupt

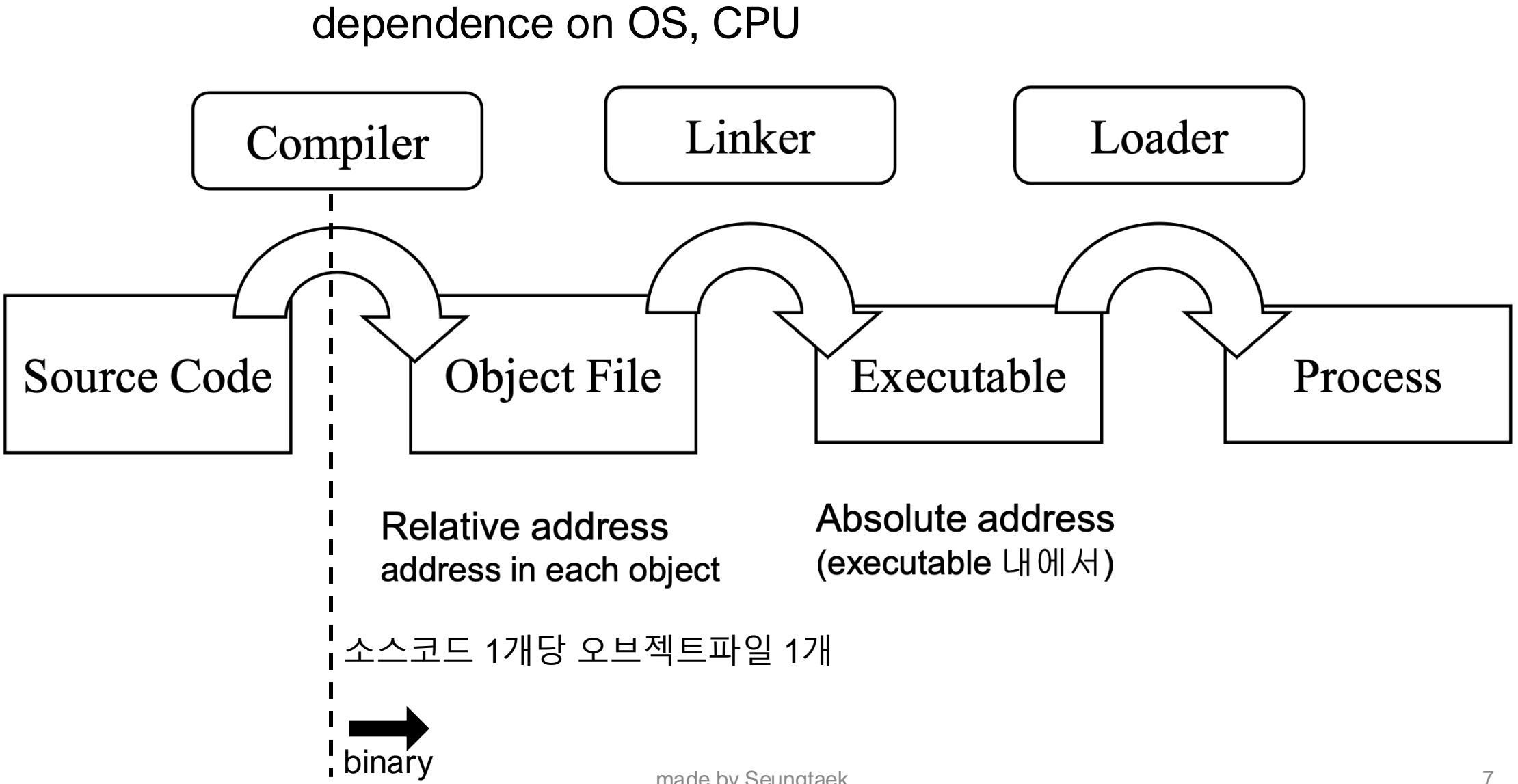


To be continue…

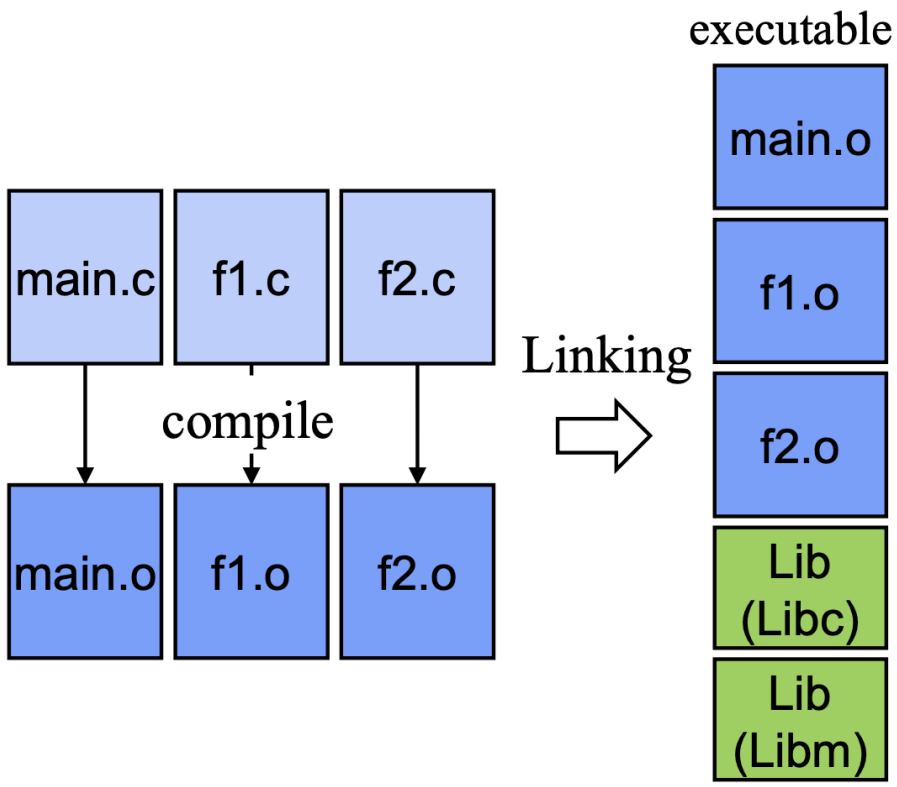
subject 1.

Program

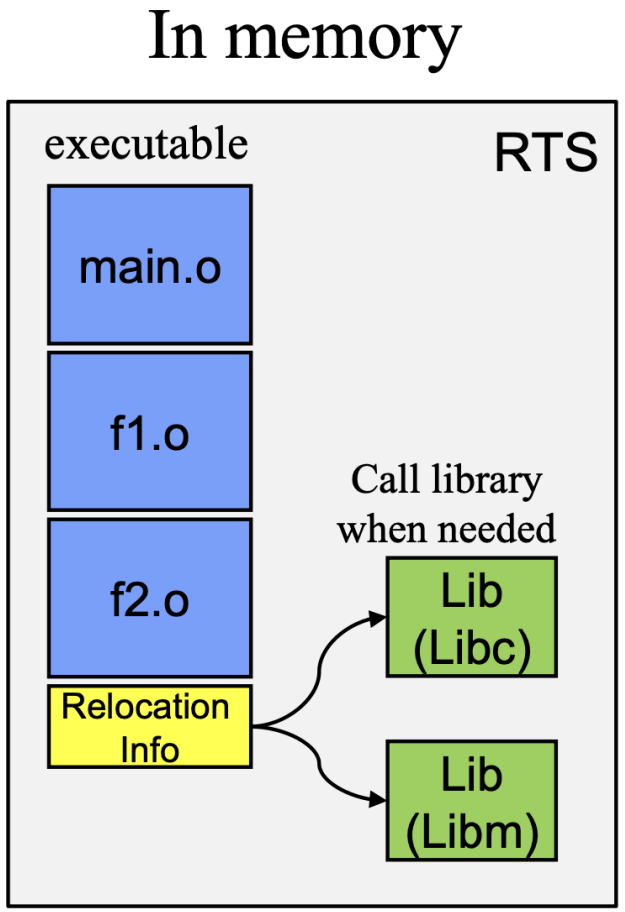
Compilation Process



Static Linking vs. Dynamic Linking



<Static Linking>
Big size executable

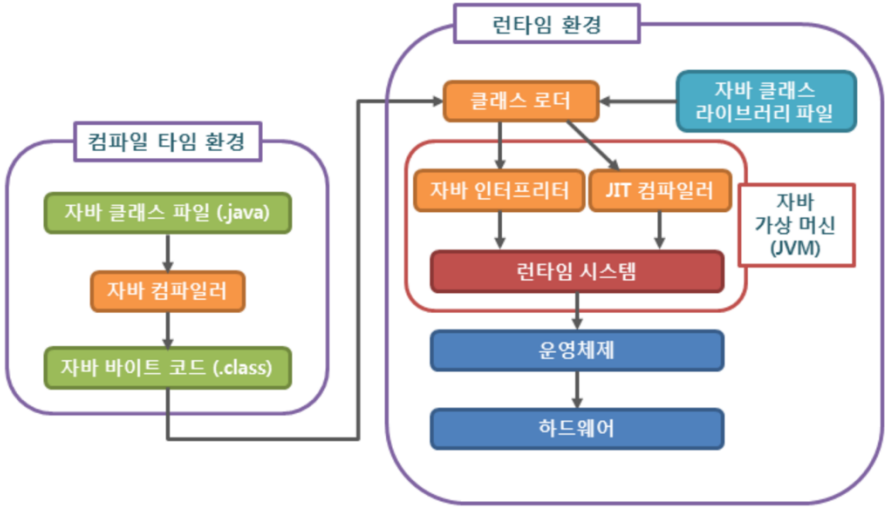


<Dynamic Linking>
ex) window **.dll**

Runtime System (RTS)

- **Programming language defines execution model**
  - RTS implements it
  - Behavior not attributable to the program
    - Process memory layout
    - Parameter passing between procedures

- **Every language has its own RTS**
  - ART: android runtime
  - Node.js: Javascript



Executable File

Windows PE (Portable Executable)

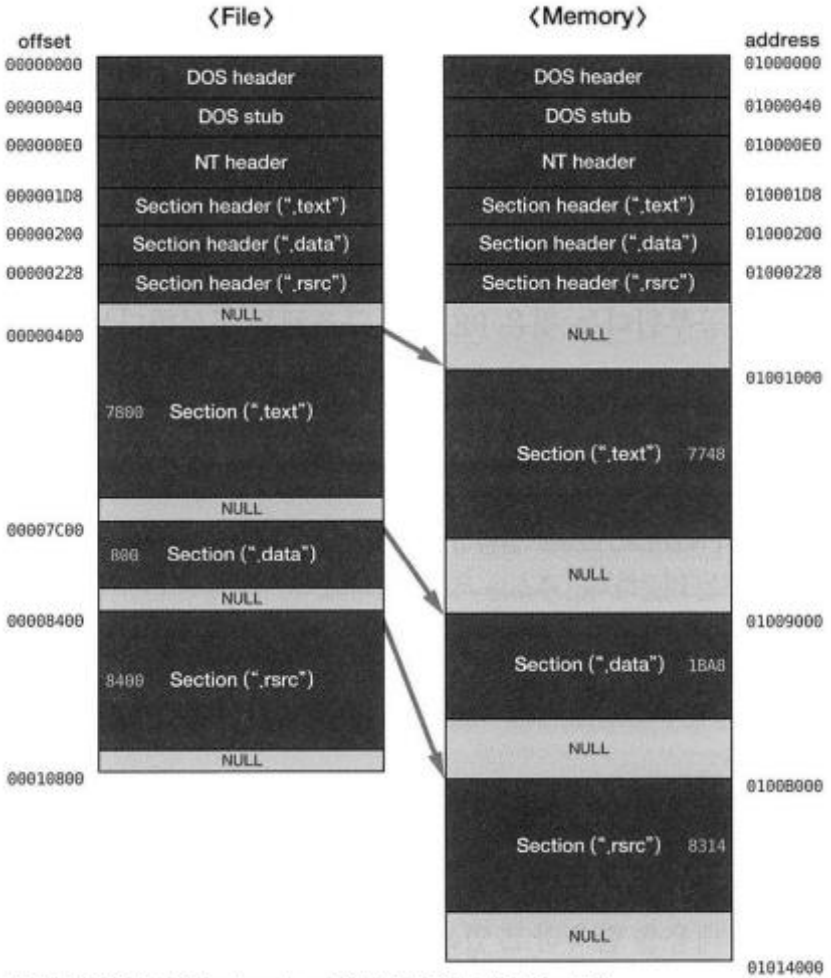
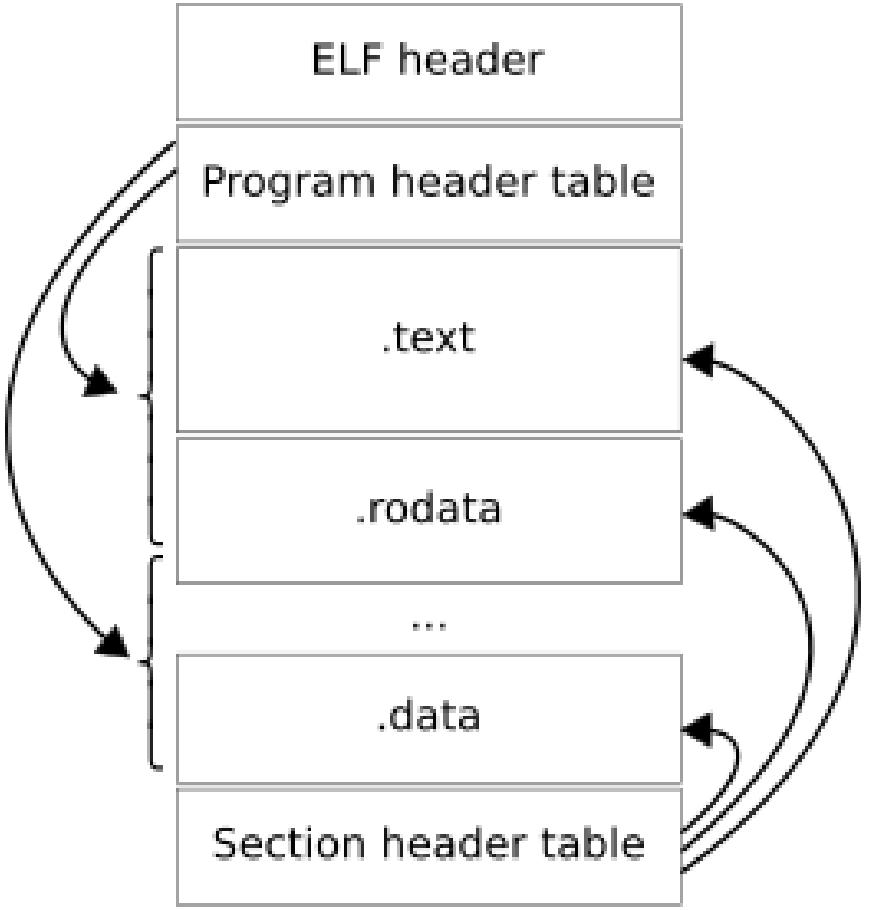
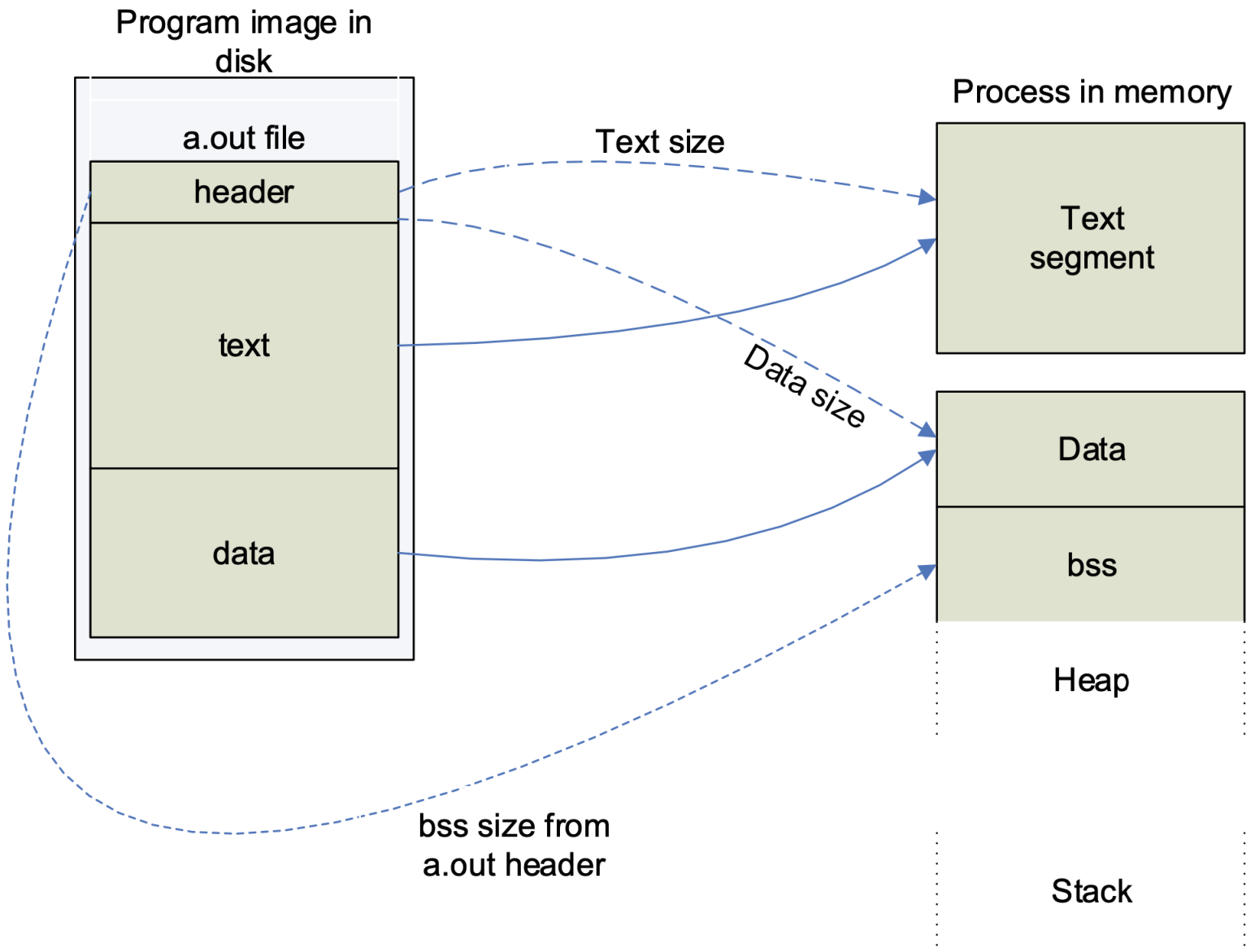


그림 13.2 PE 파일(notepad.exe)이 메모리에 로딩되는 모습

Unix ELF (Executable and Linkable Format)



Process from Program



subject 2.

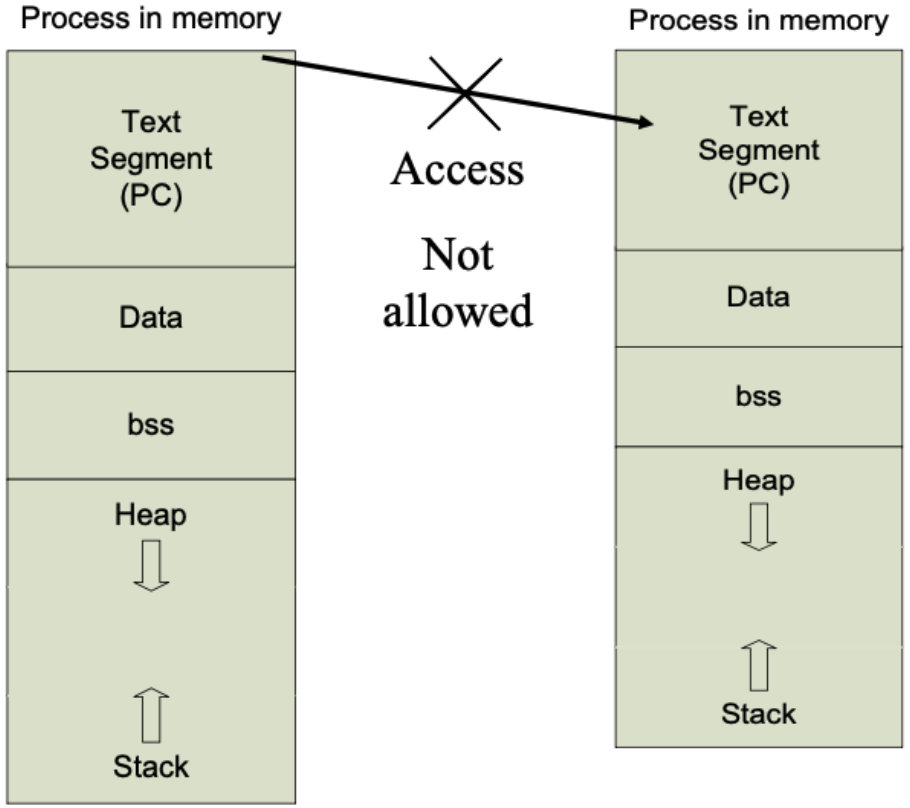
Process

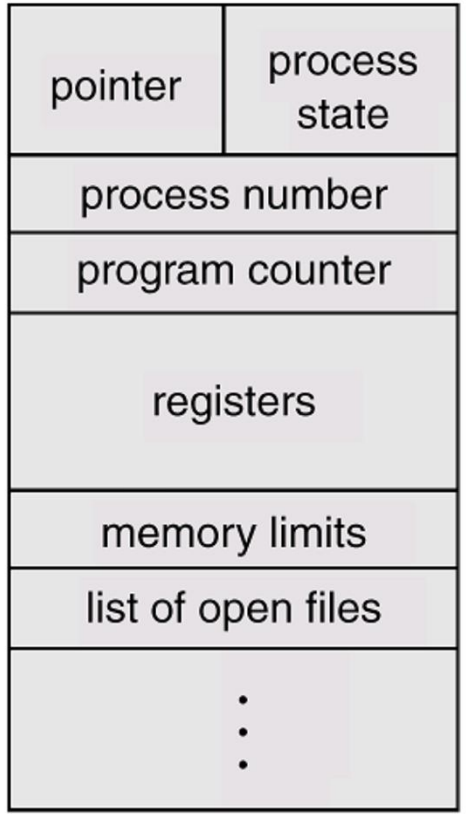
Process

실행 중인 프로그램

Process – abstraction for

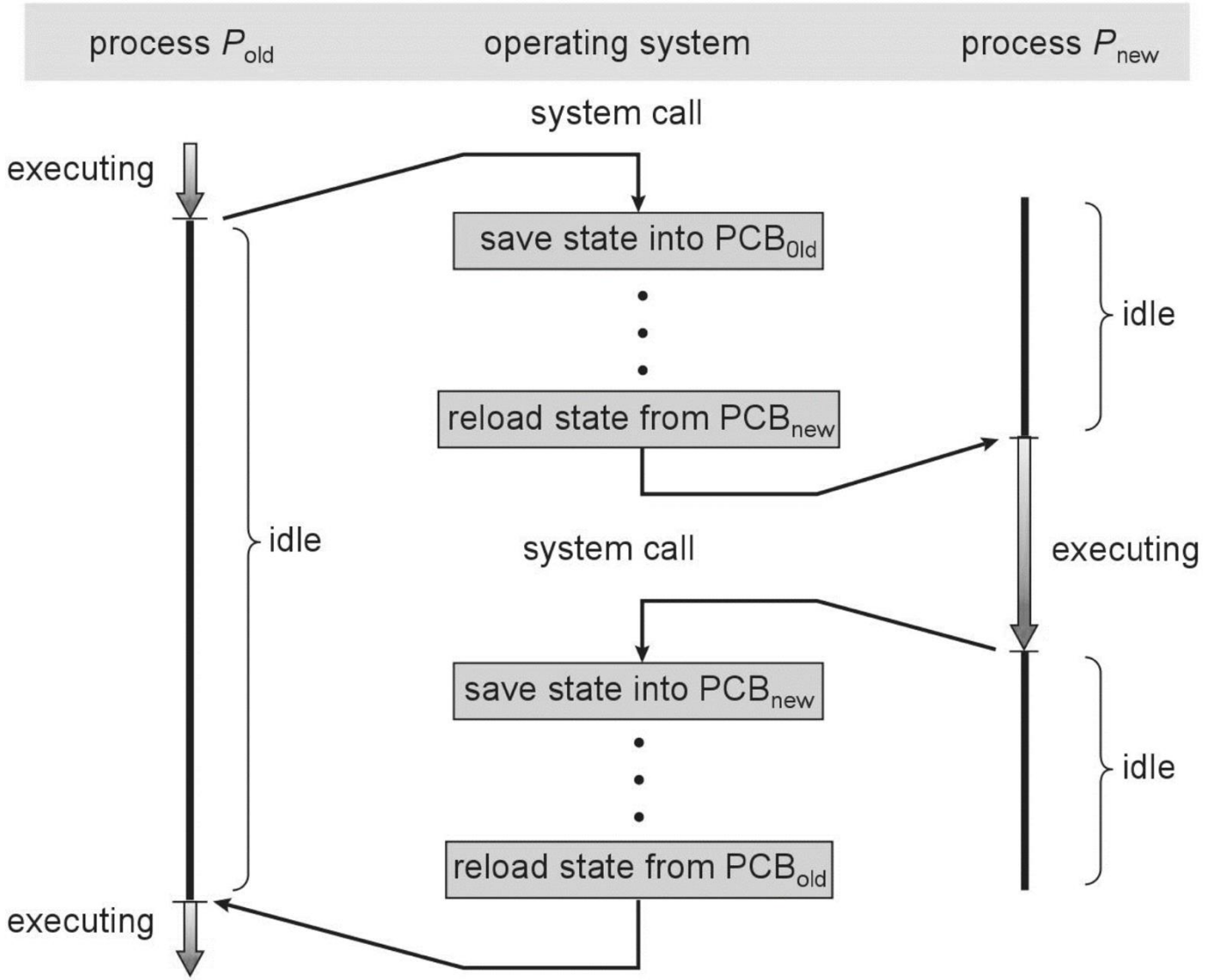
Protection domain: 서로 침범하지 못함

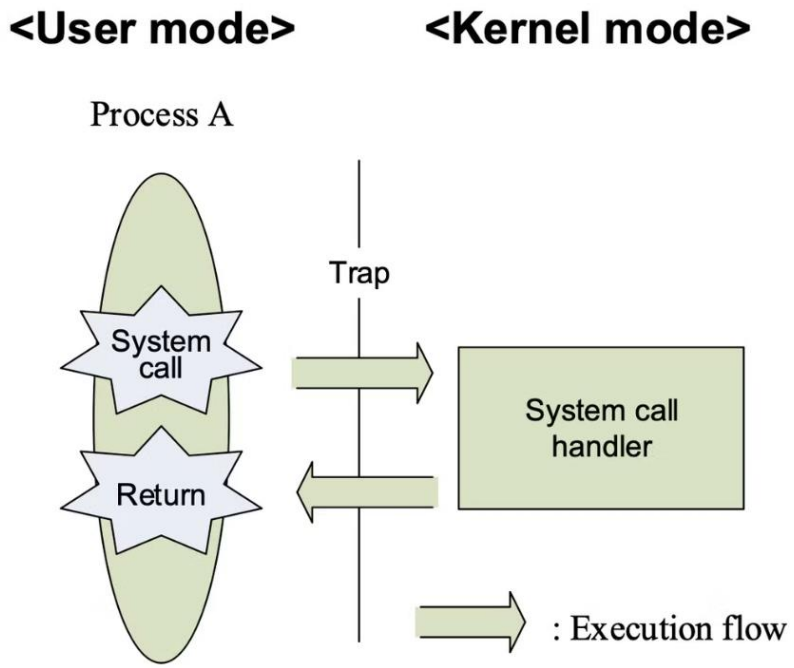




PCB(Process Control Block)

Context Switch





Context

사용자 수준 문맥(user-level context)

- 텍스트(text) 영역: 프로그램 코드 부분
- 자료(data) 영역: 프로그램 광역변수 부분
- 스택(stack) 영역: 프로그램 실행시간 스택 부분

커널 수준 문맥(Kernel-level context)

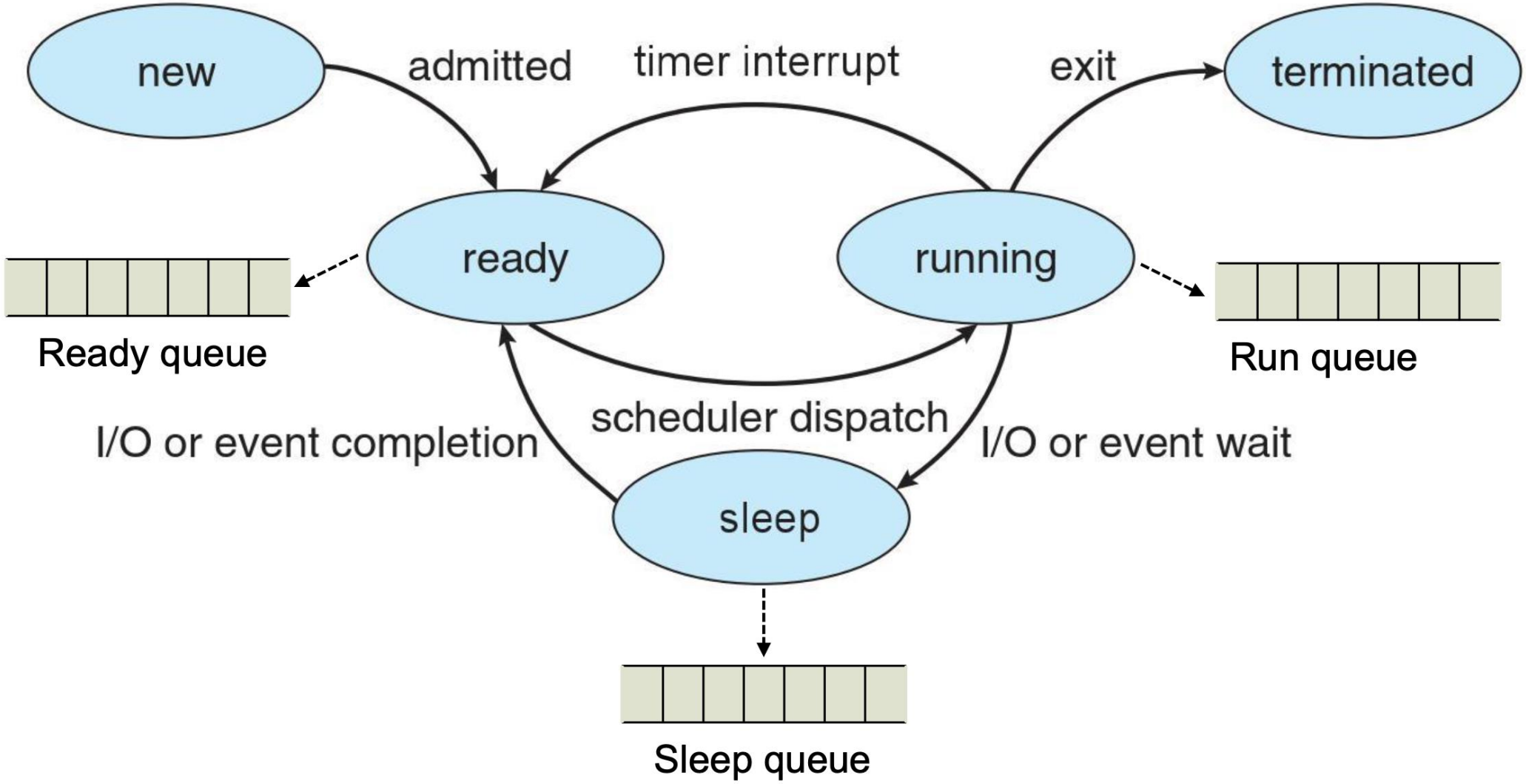
- CPU 내의 각종 **특수 레지스터**의 내용

  프로그램 카운터(PC), 스택 포인터(SP), 상태 레지스터(PSR; Program Status Register)…
- CPU 내의 각종 **범용 레지스터** 내용
- 프로세스의 현재의 각종 자원 사용 정보

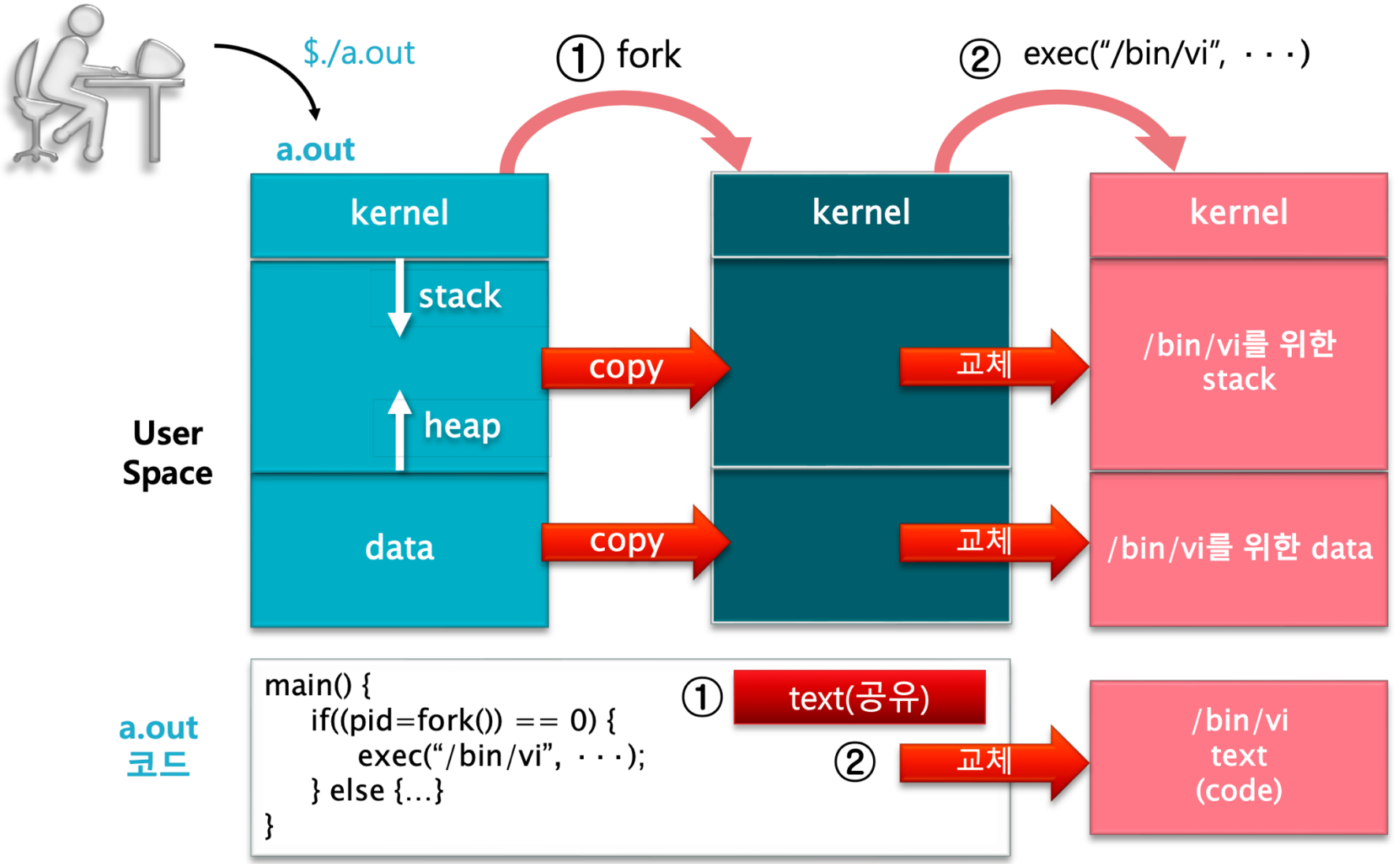
  file, 세마포어, port, ..
- 커널의 프로세스 관리 정보

  user, owner…

Process State



fork(), exec()


$./a.out
a.out
① fork
② exec("/bin/vi", · · ·)
kernel
stack
heap
data
copy
copy
kernel
교체
교체
kernel
/bin/vi를 위한 stack
/bin/vi를 위한 data
User Space
a.out 코드
main() {
if((pid=fork()) == 0) {
exec("/bin/vi", · · ·);
} else {…}
}
①
text(공유)
②
교체
/bin/vi
text
(code)

Zombie, Orphan process

Zombie process

- When process terminates, still consumes system resources
- Various tables maintained by OS (PID, exit code, …)
- Reaping (wait())
  - Parent is given exit status information
  - Kernel discards process

```
root@CT102:~# ps -e
    PID TTY          TIME CMD
      1 ?        00:00:01 systemd
```

Orphan process

- 부모 프로세스가 자식 프로세스보다 먼저 종료되면, 자식 프로세스가 부모가 없는 고아 프로세스가 된다.
- UNIX 는 고아 프로세스의 새로운 부모 프로세스를 init 프로세스로 지정
- init 프로세스는 주기적으로 wait 함수를 호출한다.

Deamon process in Linux

- 서비스의 요청에 대해 응답하기 위해 오랫동안 실행중인 백그라운드 프로세스
- typically performs tasks without requiring user interaction
- Parent PID 가 1 (init 프로세스) 이거나 다른 데몬 프로세스이다.
- Unix 에서 d 로 끝나는 프로세스 (inetd, httpd, sshd 등)

subject 3.

IPC

Interprocess Communication(IPC)

Shared memory

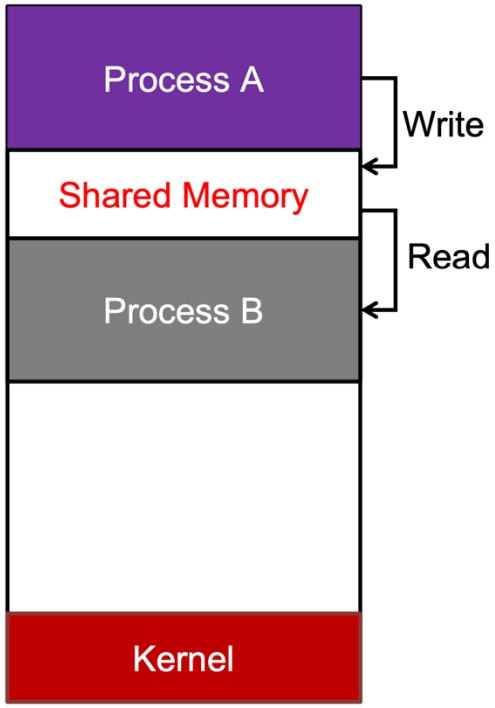
공유한 메모리 영역에 읽기/쓰기를 통해서 프로세스간 통신 수행

공유 메모리가 설정된 후부터는 **커널 관여 없음**

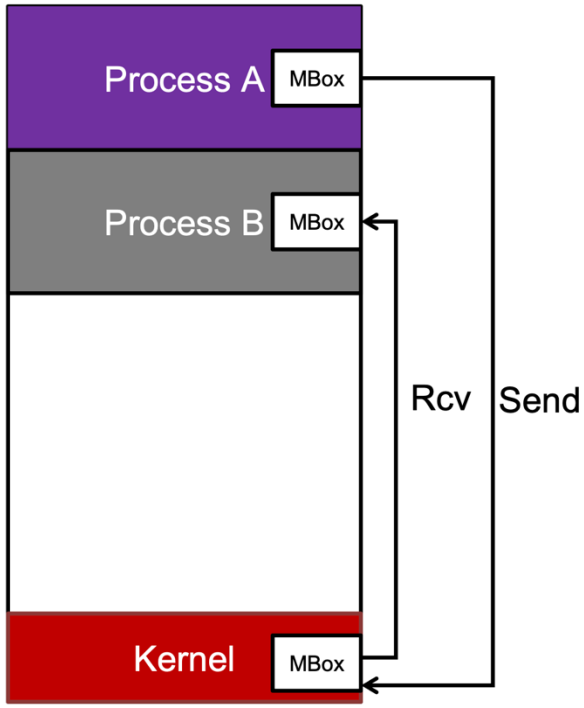
Message passing

커널을 경유해서 메시지 전송

Pipe, Message Queue, Socket



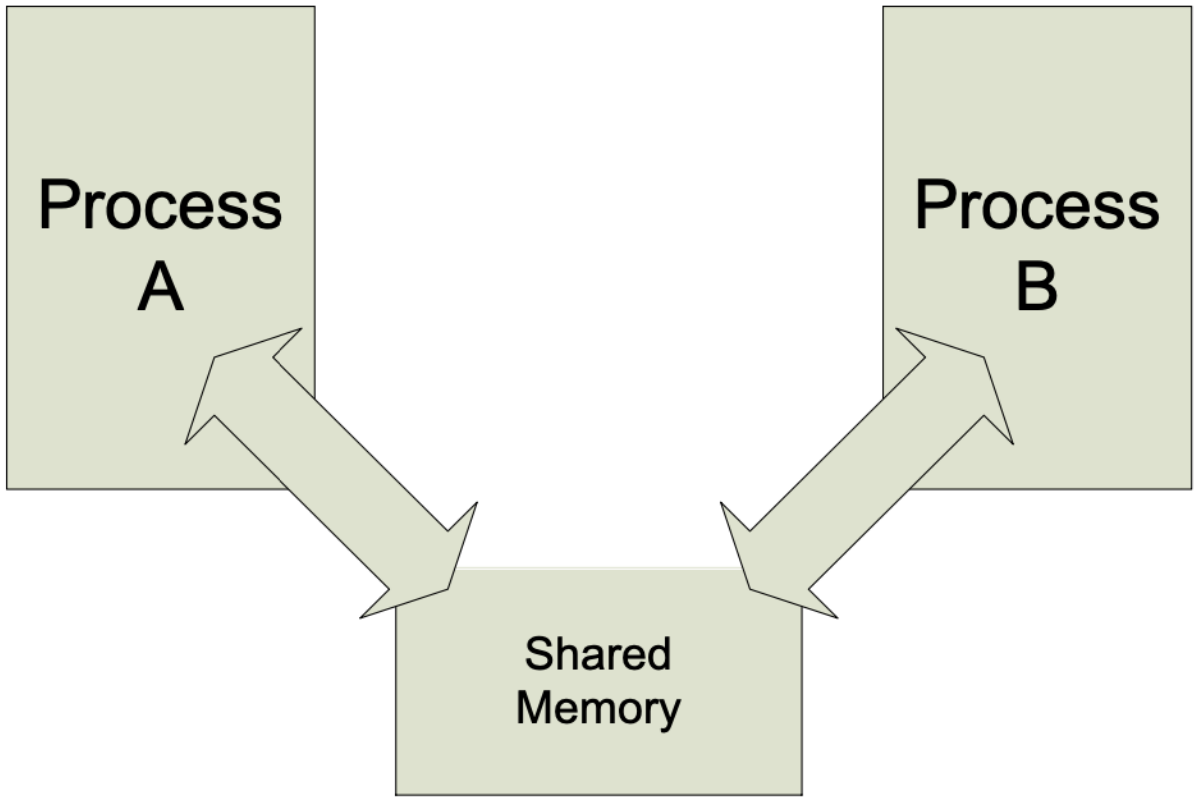
공유 메모리 기법



메시지 교환 기법

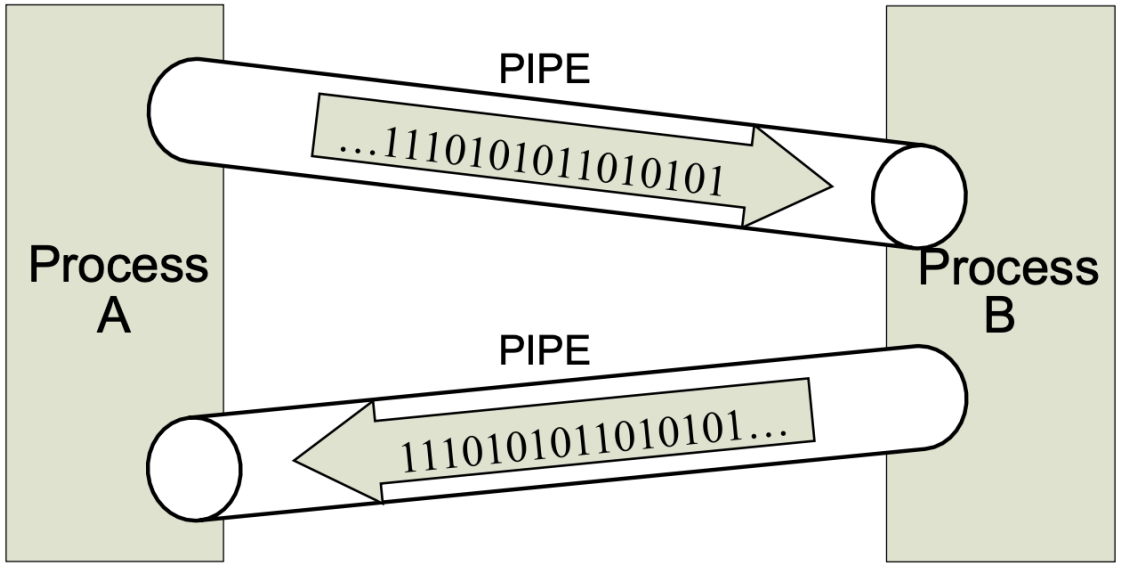
Shared Memory

- 두 개 이상의 프로세스들이 하나의 메모리 영역을 공유하여 통신을 하는 기법
- 메모리의 직접 사용으로 빠르고 자유로운 통신 가능
- 둘 이상의 프로세스가 동시에 메모리를 변경하지 않도록, 프로세스 간의 동기화가 필요함



Pipe

- 하나의 프로세스가 다른 프로세스로 데이터를 직접 전달하는 기법
- 데이터는 한 쪽 방향으로만 이동함 (half duplex)
  - 양방향 통신을 위해서는 두 개의 파이프가 필요 (full duplex)
- 보내어진 순서대로만 받음 (ordering)



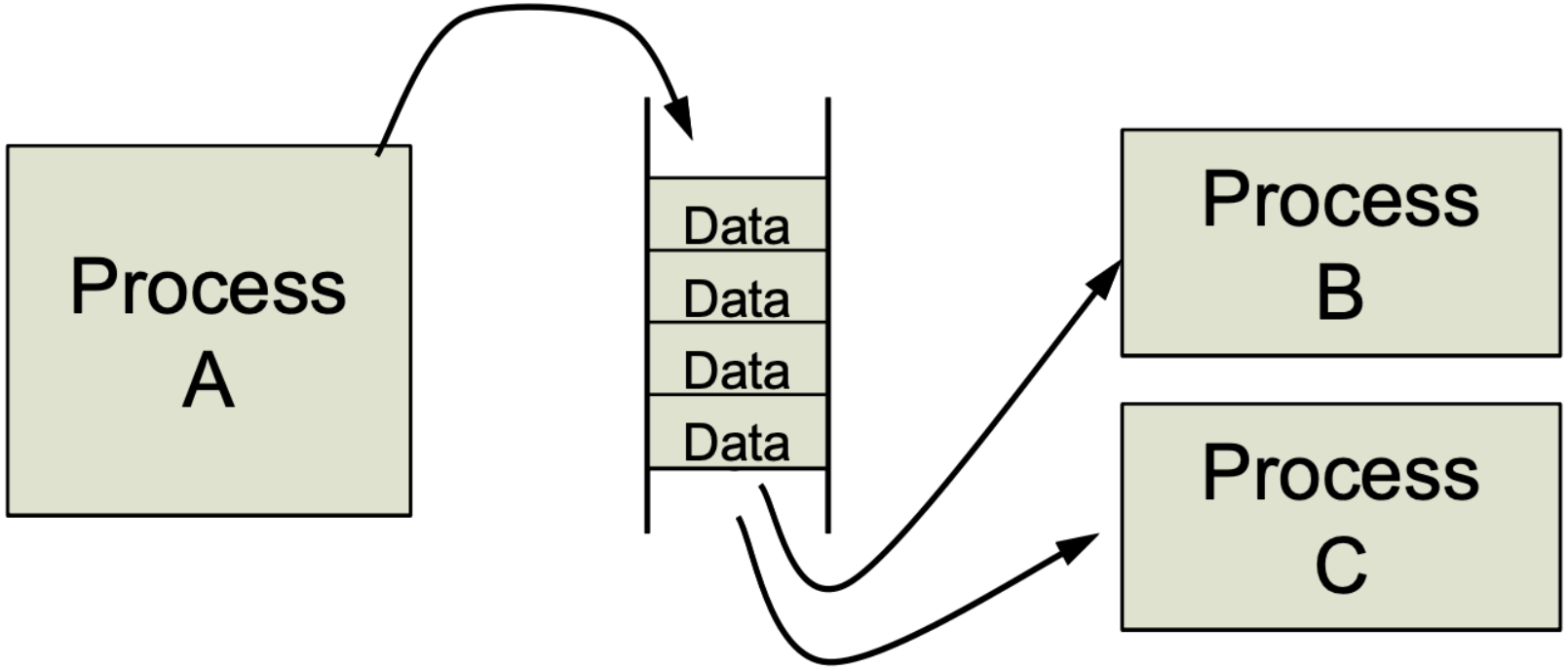
cf) ordinary pipes, named pipes

Signal

- 특정 프로세스에게 이벤트를 전달하는 기법
- 송신 프로세스 : 이벤트에 해당하는 시그널을 특정 프로세스에 보냄 (이 동작은 수신 프로세스의 상태에 무관)
- 수신 프로세스 : 시그널 종류에 따라 처리 방법을 지정함 (무시, 특정 처리 루틴(Signal handler) 수행, ..)
- 비동기적인 동작
  - Process A가 시그널을 process B에게 보내더라도, 시그널 처리는 process B가 스케줄링 되어야 가능함

```
void int_handler(int sig) {
        printf("Process %d received signal %d\n", getpid(), sig);
        _exit(0);
}
int main() {
        pid_t pid[N];
        int i, child_status;
        signal(SIGINT, int_handler);
        …
}
```

Message Queue

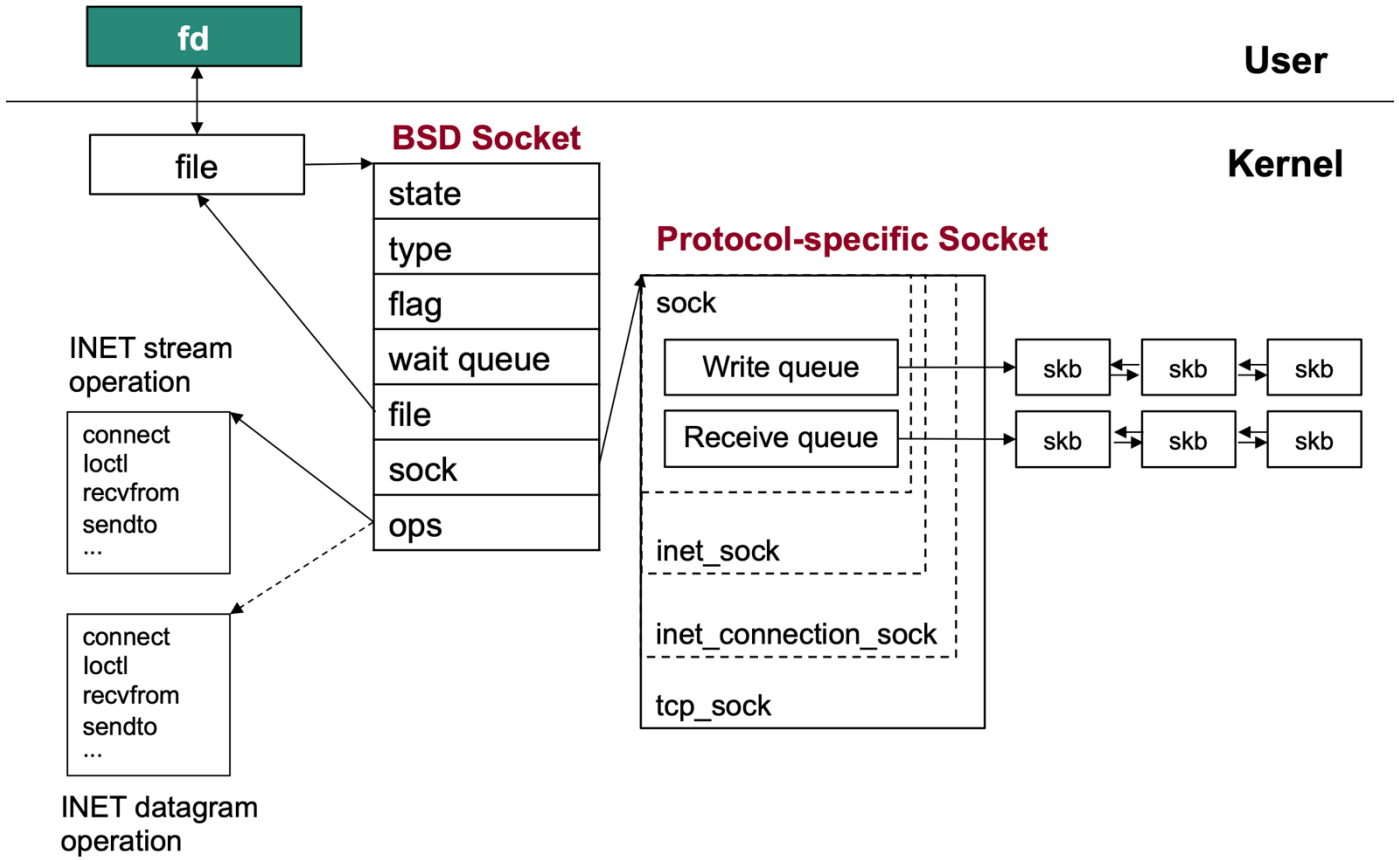


Socket

An **end-point** for Internet network “connection”
Socket is defined by the combination of **network address** and **port identifier**

Use file descriptor

- socket(2) returns fd
- This fd is used to send or receive packets

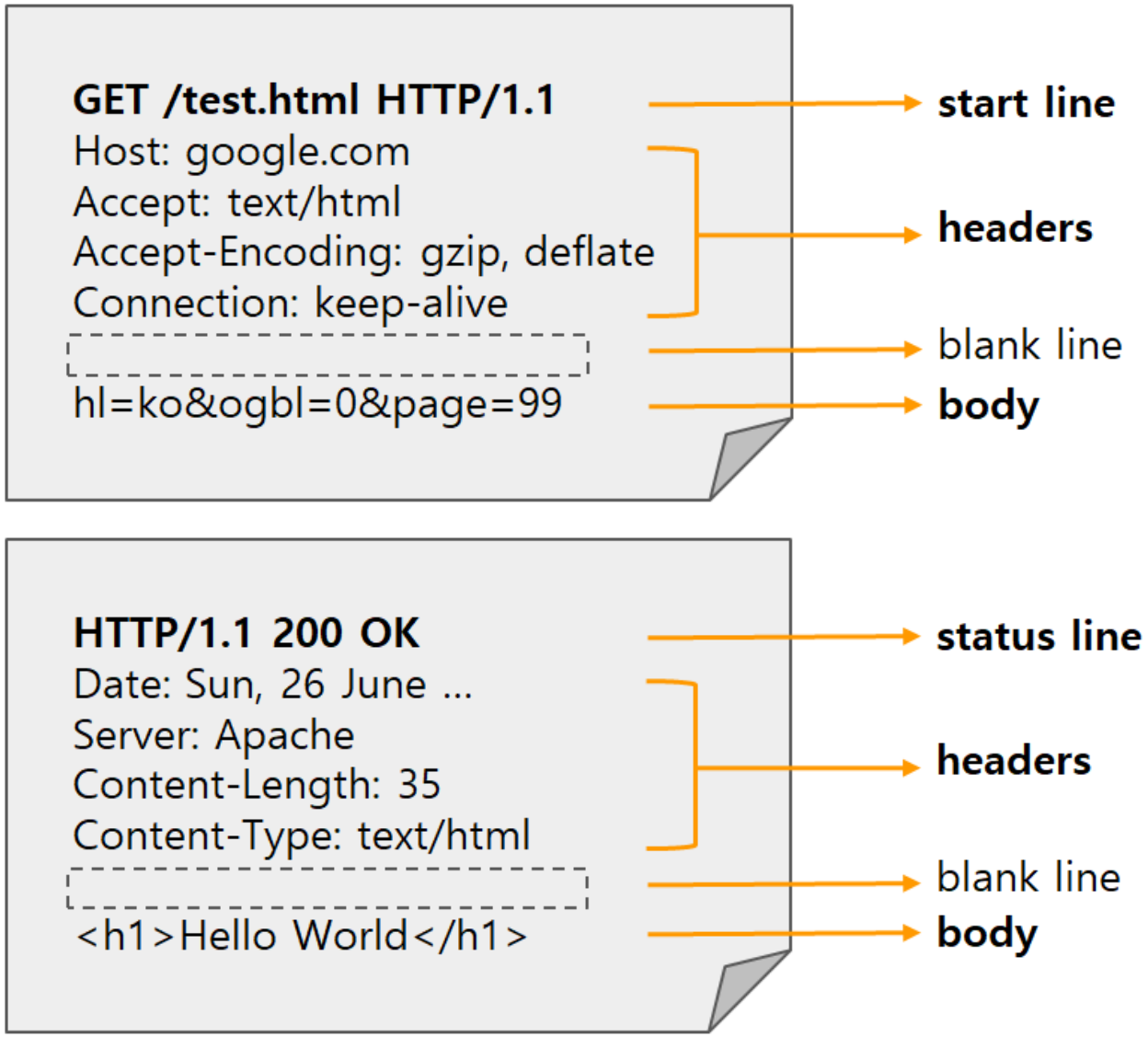


To be continued…

Practice : Socket

```
public class SocketClient {
    public static void main(String[] args) {

        final String host = "3.35.234.67";
        final String path = "/api/system-available";
        final int port = 80;

        try (Socket socket = new Socket(host, port)) { // 소켓 연결

            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);  // 응답 내용 출력
            }
        } catch (UnknownHostException e) {
            System.err.println("Cannot found the host at " + host);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Practice : Socket



HTTP Request Message

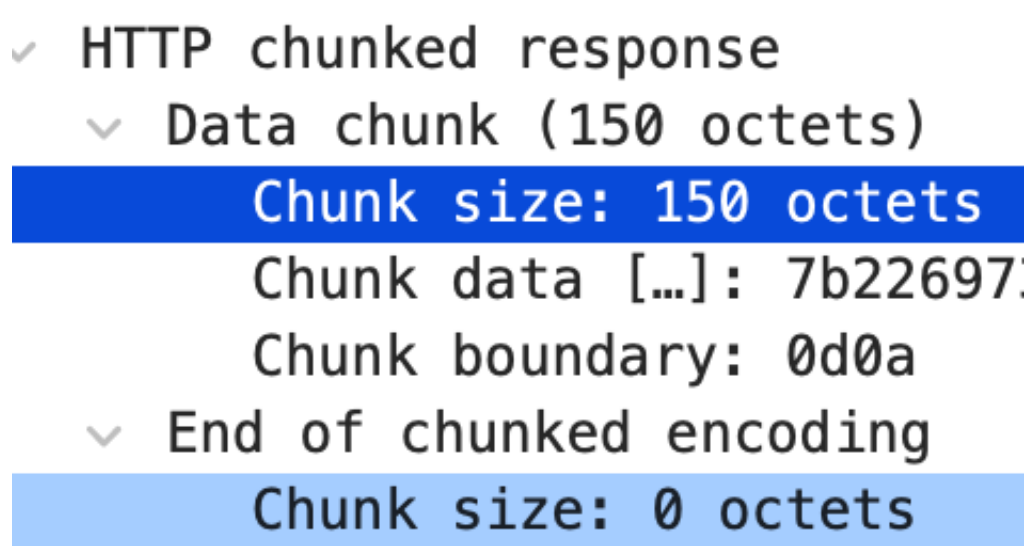
HTTP Response Message

Practice : Socket

```
public class SocketClient {
    public static void main(String[] args) {

        final String host = "3.35.234.67";
        final String path = "/api/system-available";
        final int port = 80;

        try (Socket socket = new Socket(host, port)) {
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            out.println("GET " + path + " HTTP/1.1");
            out.println("Host: " + host);
            out.println("Connection: close");
            out.println();

            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (UnknownHostException e) {
            System.err.println("Cannot found the host at " + host);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Practice : Socket

```
HTTP/1.1 200
Server: nginx/1.18.0 (Ubuntu)
Date: Mon, 23 Sep 2024 08:32:06 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: close
X-Content-Type-Options: nosniff
X-XSS-Protection: 0
Cache-Control: no-cache, no-store, m
Pragma: no-cache
Expires: 0

96
{"isSuccess":true,"response":{"syste
0
```



96(16)

= 9*16+6(10) =150(10)

https://developer.mozilla.org/ko/docs/Web/HTTP/Headers/Transfer-Encoding

subject 4.

Thread

Motivation

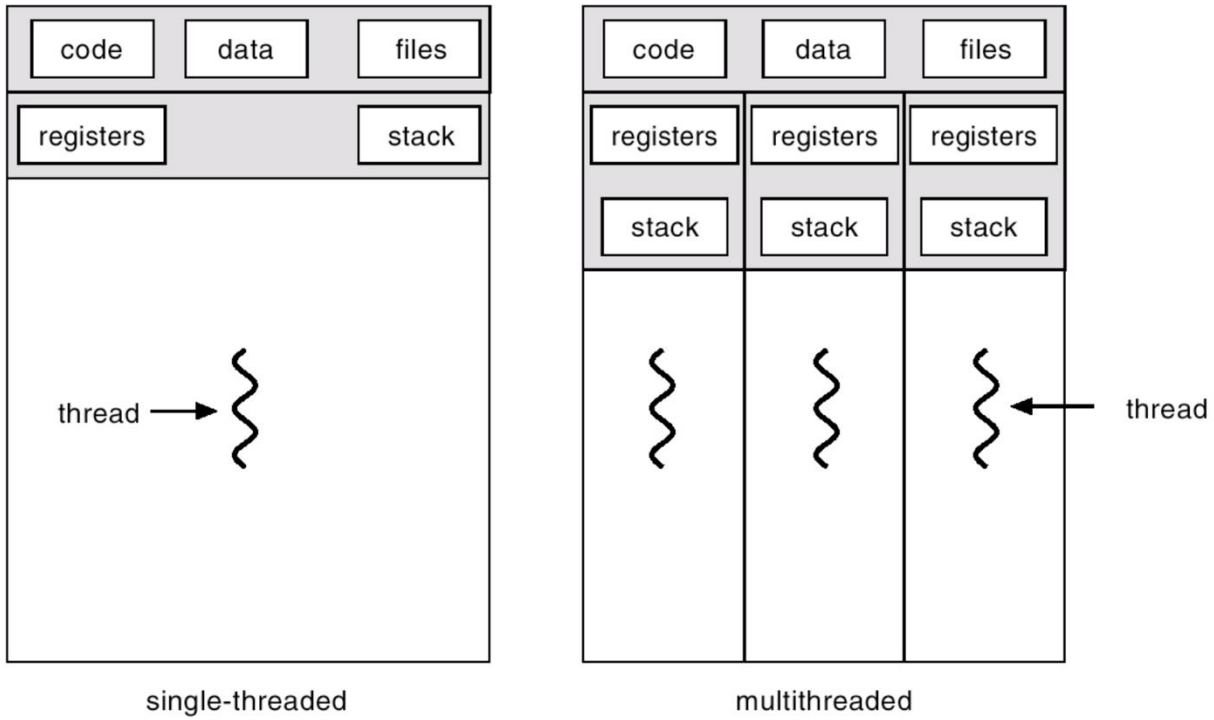
Multithread programming은 multicore 시스템에서 효율적

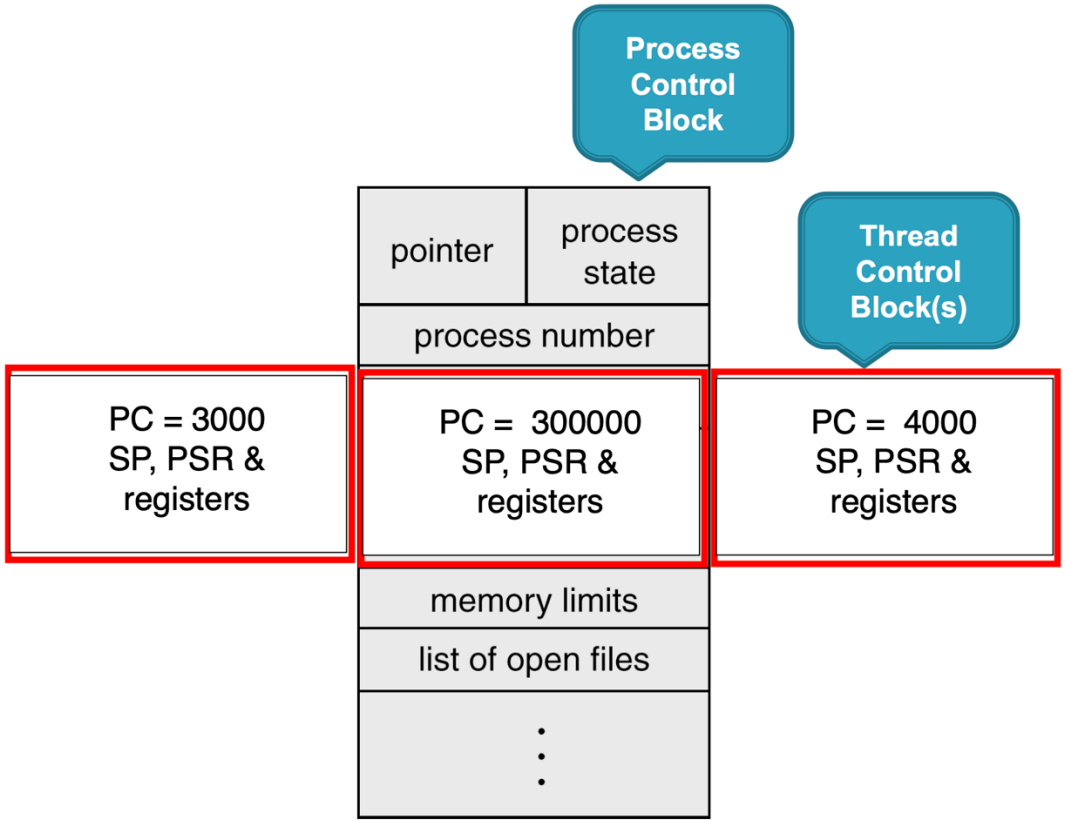
- Multicore processor는 운영체제에서 각각의 core를 하나의 프로세서로 인식하고 스케줄링
- 각각의 thread를 core에 할당하여 실행 가능
- multicore는 cache를 공유하기 때문에 data, code 등 프로세스의 자원을 공유하는 multithreaded programming에 보다 효과적임

Cooperative process와의 차이점

- Cooperative process는 프로세스간 통신(IPC)이 필요 → 비용이 많이 듦
- Process 내에서 cooperation 하는 thread로 만든다면 Process보다 적은 비용으로 cooperative process가 하는 일을 동일하게 수행 가능

Thread


code
data
files
registers
stack
thread
single-threaded
code
data
files
registers
registers
registers
stack
stack
stack
thread
multithreaded


Process Control Block
pointer
process state
process number
PC = 3000
SP, PSR & registers
PC = 300000
SP, PSR & registers
Thread Control Block(s)
PC = 4000
SP, PSR & registers
memory limits
list of open files

Process and Thread

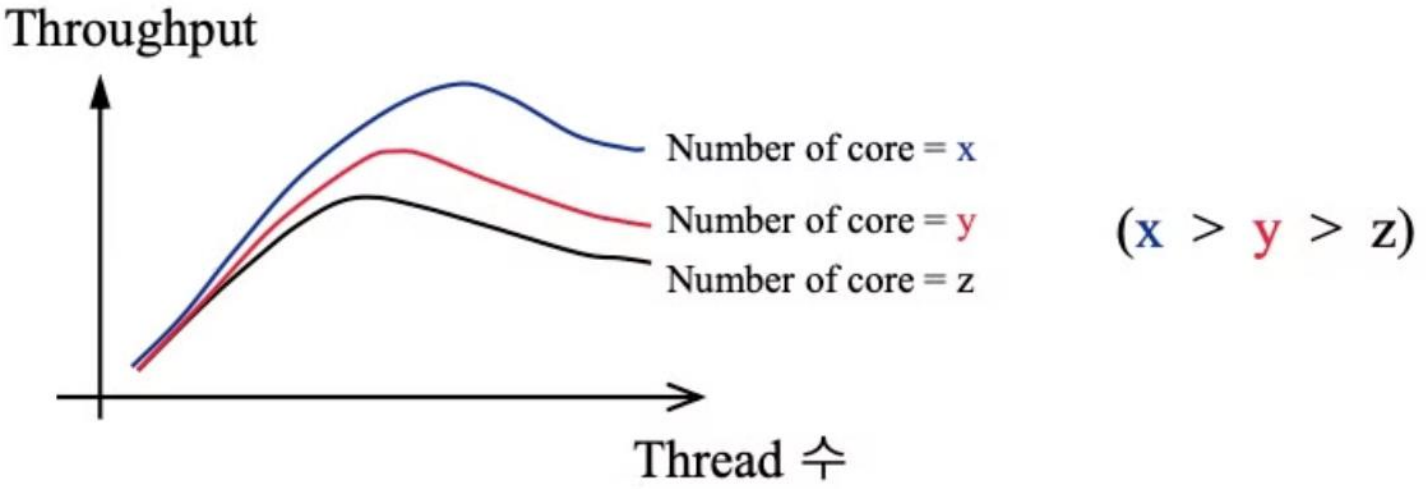
전통적 프로세스(heavy weight process)

- 서로 통신 시 운영체제가 제공하는 특수 구조체나 공유 파일을 사용(pipe, shared memory, signal, socket 등)
- fork() 시 text만 공유하고 나머지 영역 모두를 복사하여 생성하므로 시간적 오버헤드와 메모리 낭비
- 장점: 프로세스 오류가 독립적이다.

스레드(light weight process)

- 텍스트(코드)와 데이터는 공유하고 스택만 따로 갖는 형태
- 프로세스에 속한 모든 스레드는 해당 스레드의 virtual memory 공간으로 제약된다.
- 하나의 프로세스 내에 여러 개의 쓰레드 구성 가능
- 전역 변수를 쓰레드간 자료교환 수단으로 활용 (**단, 동기화 필요(mutex, semephore, …)**)
- 프로세스 생성보다 스레드 생성이 빠르고 자원 점유가 적음

Thread와 CPU utilization



(x > y > z)

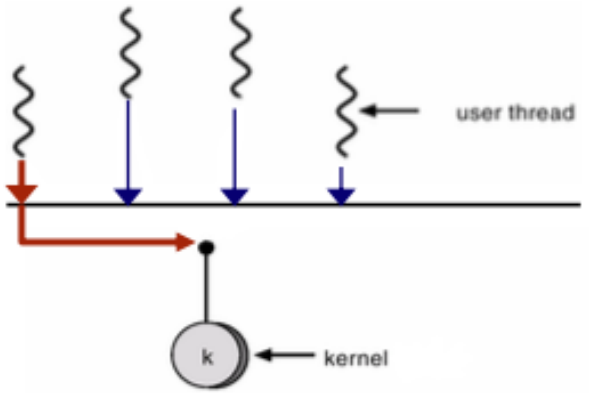
- Thread의 수가 증가할수록 CPU의 utilization이 증가
  - 임계값을 넘어가면 다시 감소 → thread switching 비용이 증가함 (with Thrashing)

- 코어의 수가 많은 시스템일수록 thread를 이용하는 게 유리
  - Multi-core 즉, 코어 수가 많을수록 한 process의 여러 thread를 parallel하게 실행

User thread, Kernel thread

User Thread

- 커널 영역 위에서 지원되며 일반적으로 user level의 라이브러리를 통해 구현됨
- 라이브러리에서 thread를 생성, 스케줄링과 관련된 관리 수행
- 장점 : thread의 생성, 관리 속도가 빠르다.
- **단점** : 여러 개의 user thread 중에서 하나의 thread가 block이 된다면 나머지 모든 thread 역시 block 된다.
  - Kernel은 여러 개의 user thread들을 하나의 process로 간주하기 때문
  - 커널이 볼 때는 스레드가 1개인지 여러 개인지 모른다.



User thread, Kernel thread

Kernel Thread

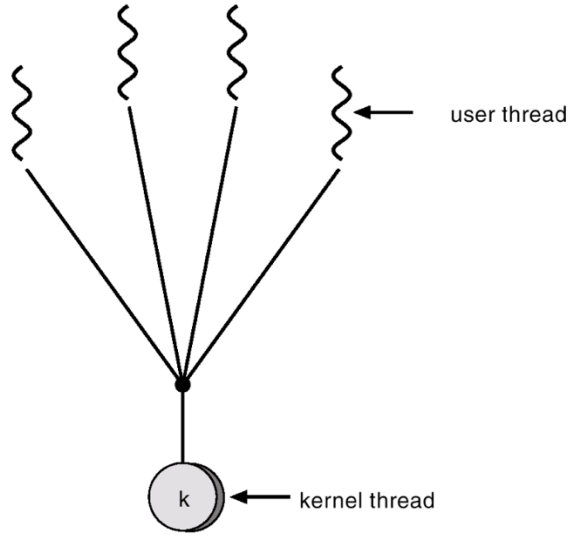
- 운영체제에서 thread를 지원 (커널 영역에서 thread의 생성, 스케줄링 등을 관리)
- **장점**
  - Thread가 system call을 호출하여 block이 되면, kernel은 다른 thread를 실행함으로써 전체적인 thread blocking이 없음
  - Multiprocessor 환경에서 커널이 여러 개의 thread를 다른 processor에 할당할 수 있음 (physical parallelism)
- 단점
  - User thread보다 생성 및 관리가 느림

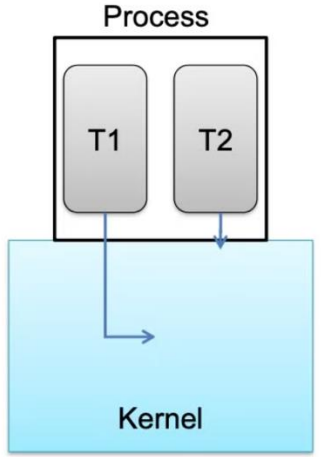
Mapping User-Kernel thread : Many-to-One

- Thread 관리는 user level에서 이루어짐
- 여러 개의 user level thread들이 하나의 kernel thread로 매핑됨
- Kernel thread를 지원하지 못하는 시스템에서 사용됨 (임베디드 커널 등)

한계점

- 한번에 하나의 thread만 커널에 접근 가능
  - 하나의 thread가 커널에 접근(system call)하면 나머지 thread들은 대기해야 함
- Kernel의 입장에서 하나의 thread이기 때문에 multicore이더라도 여러 개의 코어에서 동시에 수행 불가능

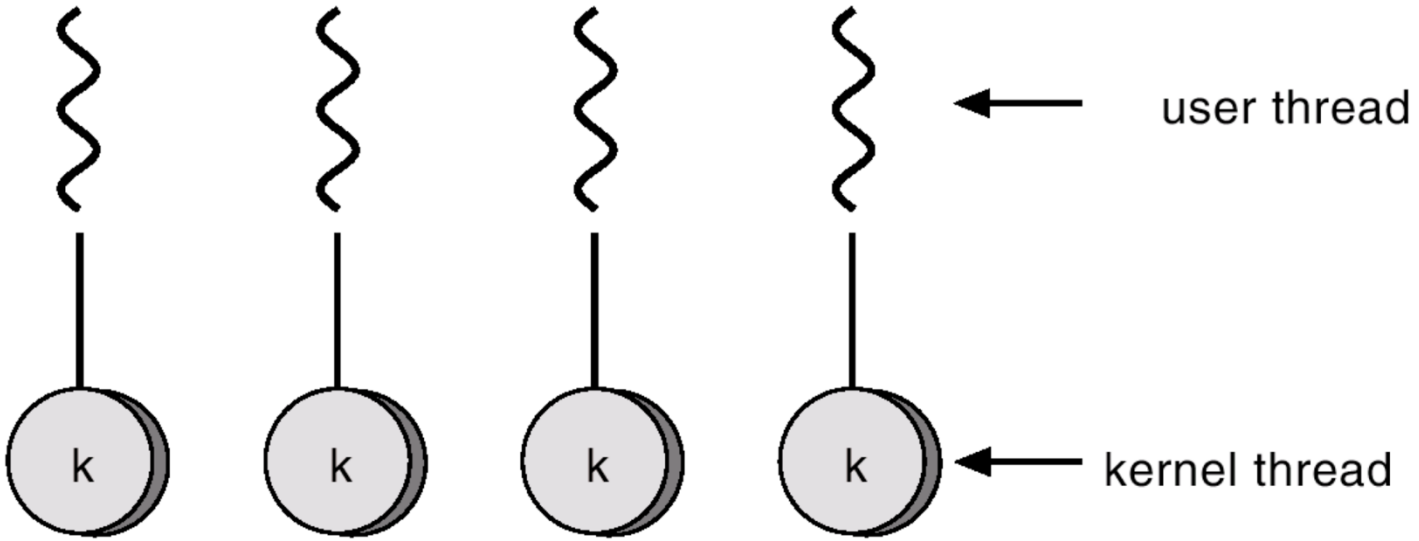




T1이 system call을 수행하는 동안
T2는 커널 진입이 불가능

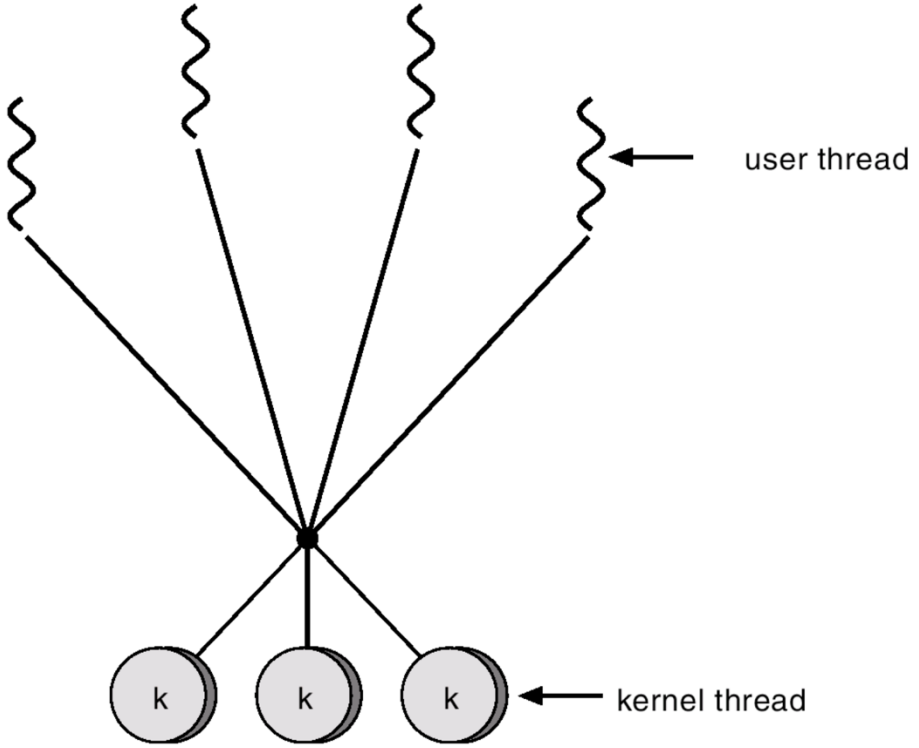
Mapping User-Kernel thread : One-to-One

- 각각의 user thread를 kernel thread로 매핑
- system call 호출 시 다른 thread들이 block되는 문제를 해결
- 여러 개의 thread를 multicore에서 동시적으로 수행 할 수 있음
- 한계점
  - Kernel thread도 한정된 자원이기 때문에 무한정으로 생성 할 수 없음
  - Thread를 생성, 사용하려 할 때 그 개수에 대한 고려가 필요



Mapping User-Kernel thread : Many-to-Many

- Kernel thread는 생성된 user thread와 같거나 적은 수 만큼만 생성이 되어 적절히 스케줄링
- One-to-One처럼 사용할 thread의 수에 대해 고민할 필요 없음
- Many-to-One처럼 thread가 system call을 사용할 경우, 다른 thread들이 block되는 현상에 대해 걱정할 필요 없
- ex) KVM의 guestOS에 vCPU 할당, Goroutines



Implementation

POSIX Pthread: interoperability를 위한 표준
- IEEE 1003.1c: pthread_create()

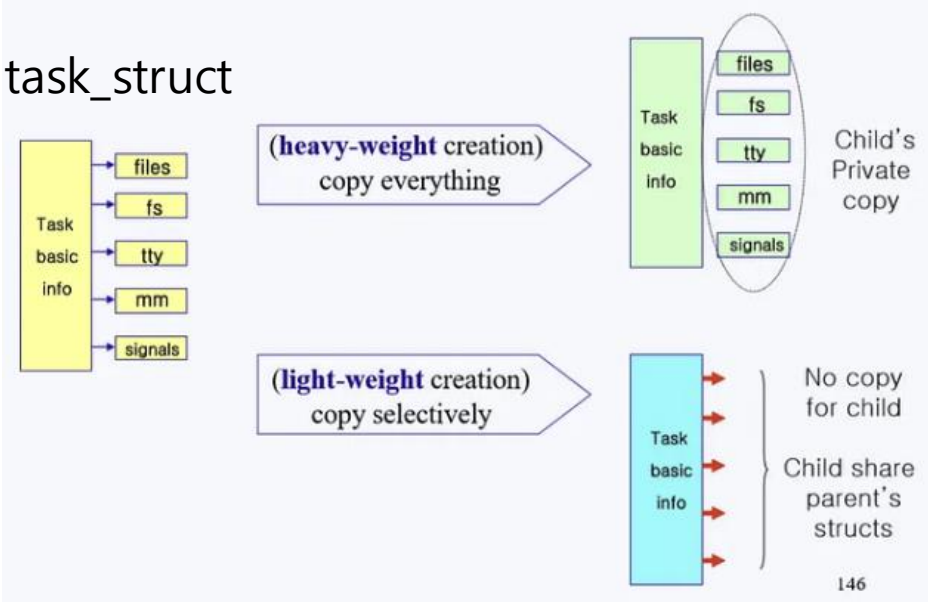
Windows threads API
- Via system call: CreateThread()

Linux threads
- Via system call: clone()

Java threads
- Thread class
- 리눅스라면, 내부적으로 clone을 호출

Thread in Linux

Thread implemented as process



clone()이 호출될 때 전달되는 플래그의 일부

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

kernel thread exist

run solely in kernel-space

schedulable and preemptible

코드 보면, 사방이 mutex 임

CF) Hardware Thread

	인텔 i7-14700K	라이젠 9700X	애플 M3 Pro
코어	P8+E12	8	P6+E6
스레드	16+12 = 28	16	6+6 = 12

SMT(Simultaneous Multi-Threading) : 동시 멀티스레딩

예전에는 4-Way, 8-Way도 사용됐지만, 현재는 2-Way가 가장 널리 사용됨

인텔의 상표명인 '**하이퍼스레딩(HT)**'이 대명사처럼 사용됨

정작 인텔은 HT를 조금씩 빼는..

subject 5.

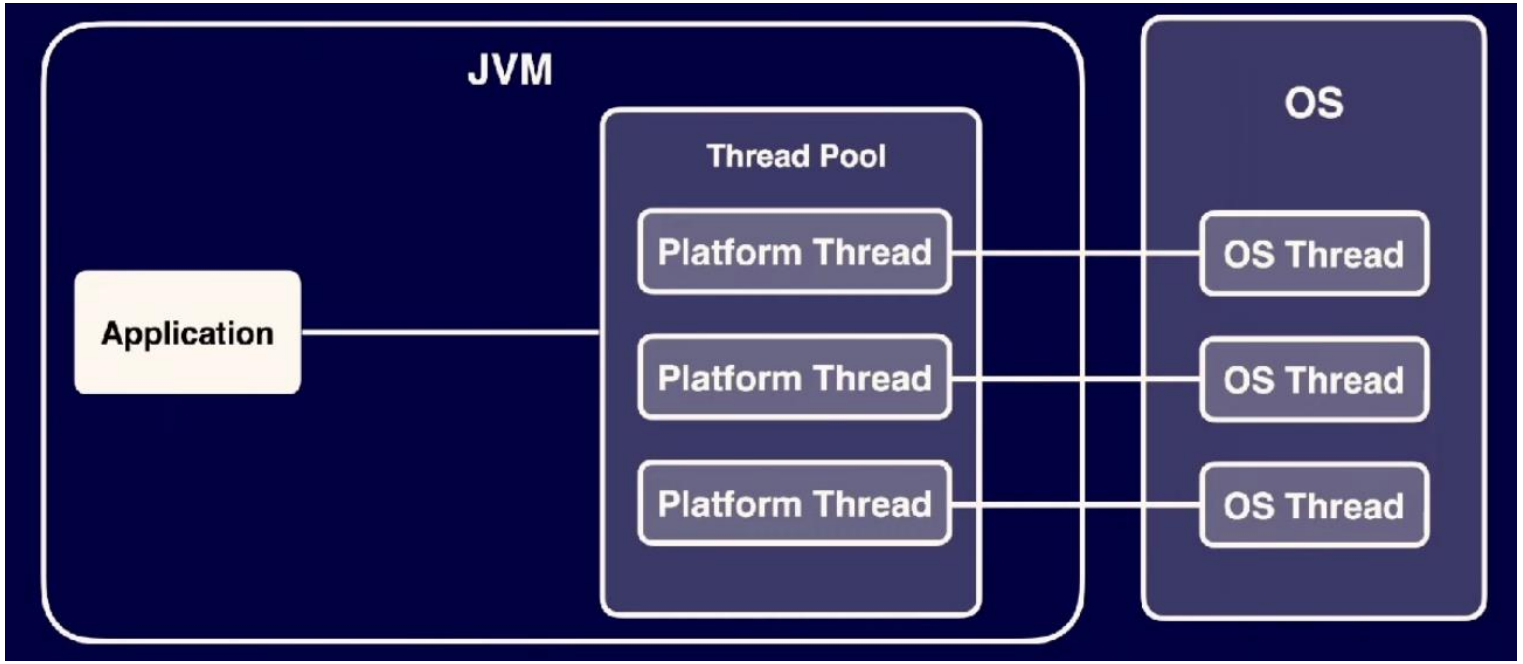
Java Thread

Java Thread

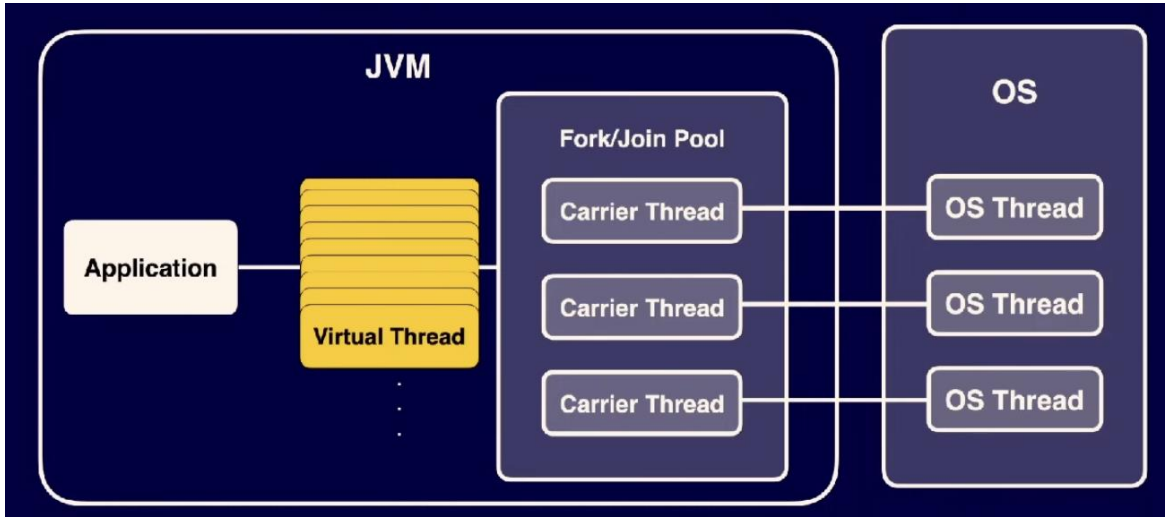
Java의 Thread는 OS Thread를 Wrapping 한 것 (Platform Thread)

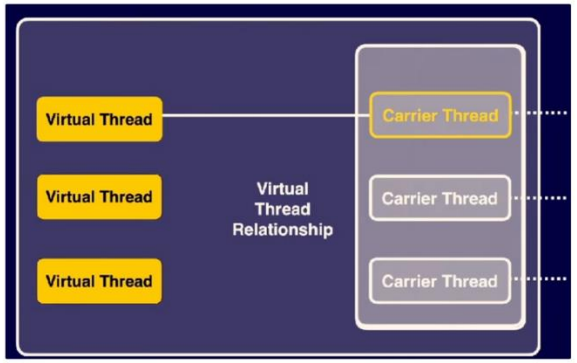
OS Thread는 생성 개수가 제한적이고 생성, 유지하는 비용이 비싸다 -> Thread Pool 사용

```java
// java.lang.Thread.java

private native void start0();

public void start() {
    // ...

    start0();

    // ...
}
```

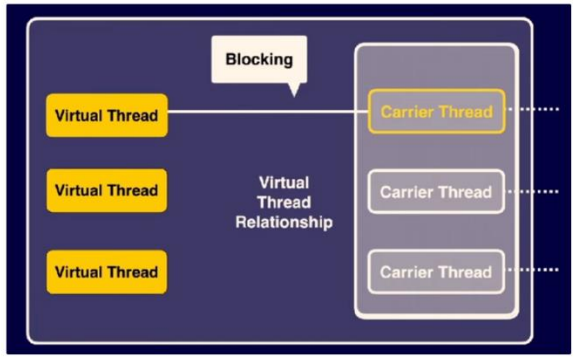


Java Virtual Thread

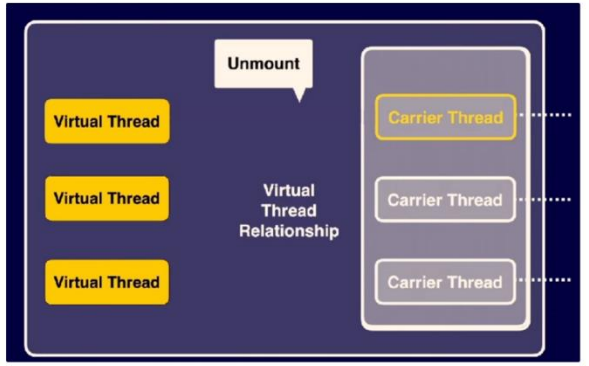




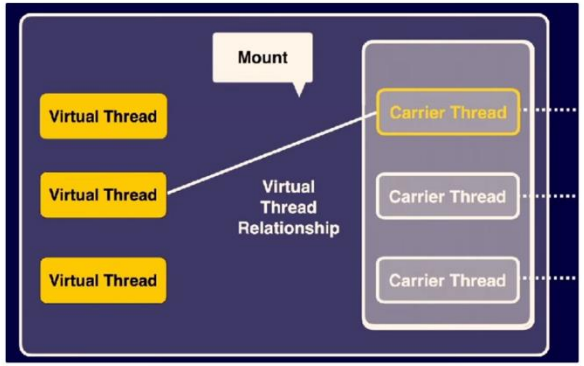
1



2



3



4

Java Virtual Thread

1. Thread Local 사용시 Heap 메모리 주의
2. synchronized 을 사용하면 Carrier Thread가 block 된다. (pinning)
   - 대신 ReentrantLock을 사용
3. **Overwhelming 주의**
4. I/O Blocking이 발생하는 경우에 적합함. CPU Intensive에는 적합하지 않다.
5. 항상 Daemon 스레드로 동작한다.
6. Platform thread 개수 (https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Thread.html)

System properties

System property	Description
`jdk.virtualThreadScheduler.parallelism`	The number of platform threads available for scheduling virtual threads. It defaults to the number of available processors.
`jdk.virtualThreadScheduler.maxPoolSize`	The maximum number of platform threads available to the scheduler. It defaults to 256.

Reference

Operating System (korea university, 유혁)

Operating System Concepts (10/E, Silberschatz)

Computer Systems A Programmer’s Perspective (3/E, Randal E. Bryan)

Computer Organization And Design (6/E, David A. Patterson)

커리큘럼

날짜	내용
9월 2일	OT
9월 9일	시스템콜, 인터럽트
9월 16일	추석
9월 23일	프로세스, 스레드
9월 30일	**Network in Linux**
10월 7일	**중간고사**
10월 14일	**중간고사**

날짜	내용
10월 21일	**중간고사**
10월 28일	**동기화, 교착상태**
11월 4일	**가상메모리, 캐시**
11월 11일	스케줄러, 파일시스템
11월 18일	동기/비동기, 블로킹/논블로킹
11월 25일	자바(JVM, GC), Spring