프로스세간 커뮤니케이션

- 각 IPC 기법 개념 이해하기

이준희 강사 (dream@fun-coding.org)

다양한 IPC 기법

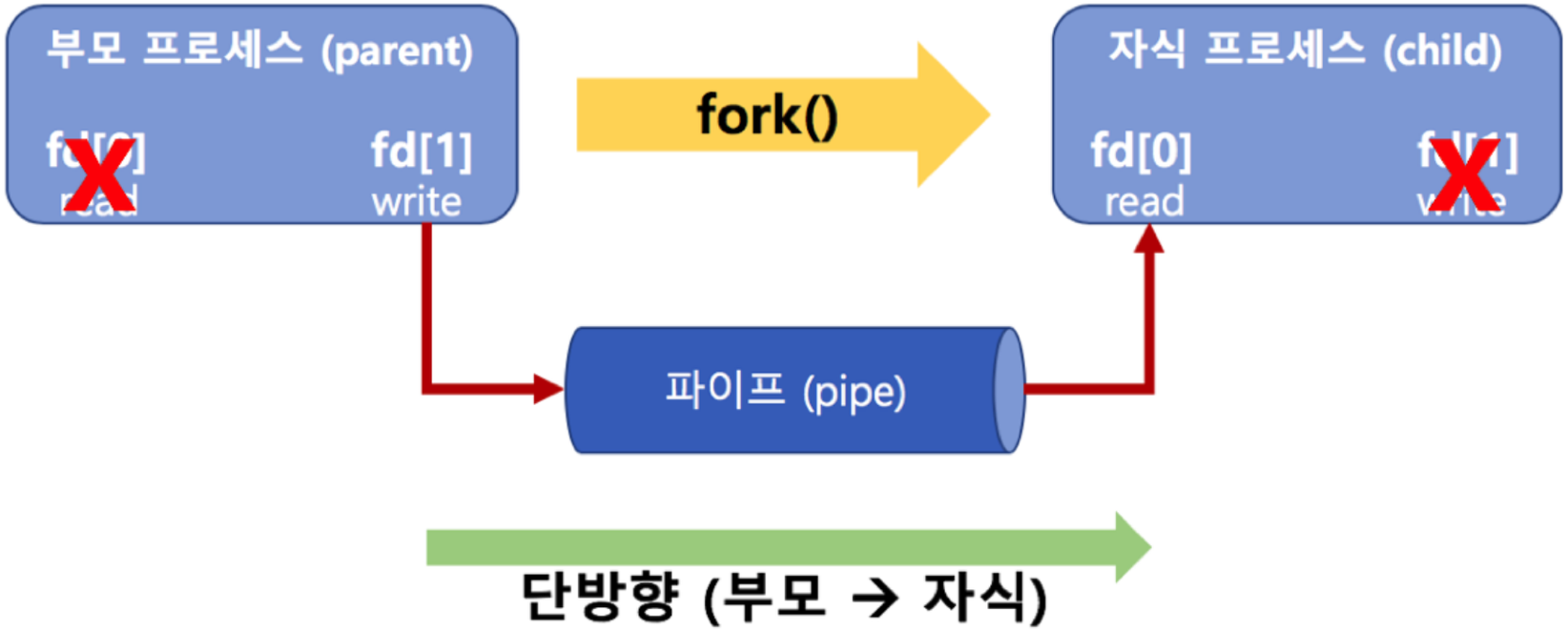
- IPC: InterProcess Communcation

1. file 사용
2. Message Queue
3. Shared Memory
4. Pipe
5. Signal
6. Semaphore
7. Socket

...

파이프

- pipe (파이프)
  - 기본 파이프는 단방향 통신
  - fork() 로 자식 프로세스 만들었을 때, 부모와 자식간의 통신



파이프 코드 예제

```
char* msg = "Hello Child Process!";
int main()
{
    char buf[255];
    int fd[2], pid, nbytes;
    if (pipe(fd) < 0)  // pipe(fd) 로 파이프 생성
        exit(1);
    pid = fork(); // 이 함수 실행 다음 코드부터 부모/자식 프로세스로 나뉘어짐
    if (pid > 0) {  // 부모 프로세스는 pid에 실제 프로세스 ID가 들어감
        write(fd[1], msg, MSGSIZE); //fd[1]에 씁니다.
        exit(0);
    }
    else {  // 자식 프로세스는 pid가 0이 들어감
        nbytes = read(fd[0], buf, MSGSIZE); // fd[0]으로 읽음
        printf("%d %s\n", nbytes, buf);
        exit(0);
    }
    return 0;
}
```

메시지 큐(message queue)

- 큐니까, 기본은 FIFO 정책으로 데이터 전송

* 출처: http://www.stoimen.com/blog/2012/06/05/computer-algorithms-stack-and-queue-data-structure/

메시지 큐 코드 예제

- A 프로세스

```
msqid = msgget(key, msgflg) // key는 1234, msgflg는 옵션
msgsnd(msqid, &sbuf, buf_length, IPC_NOWAIT)
```

- B 프로세스

```
msqid = msgget(key, msgflg) // key는 동일하게 1234로 해야 해당 큐의 msgid를 얻을 수 있음
msgrcv(msqid, &rbuf, MSGSZ, 1, 0)
```

파이프와 메시지 큐

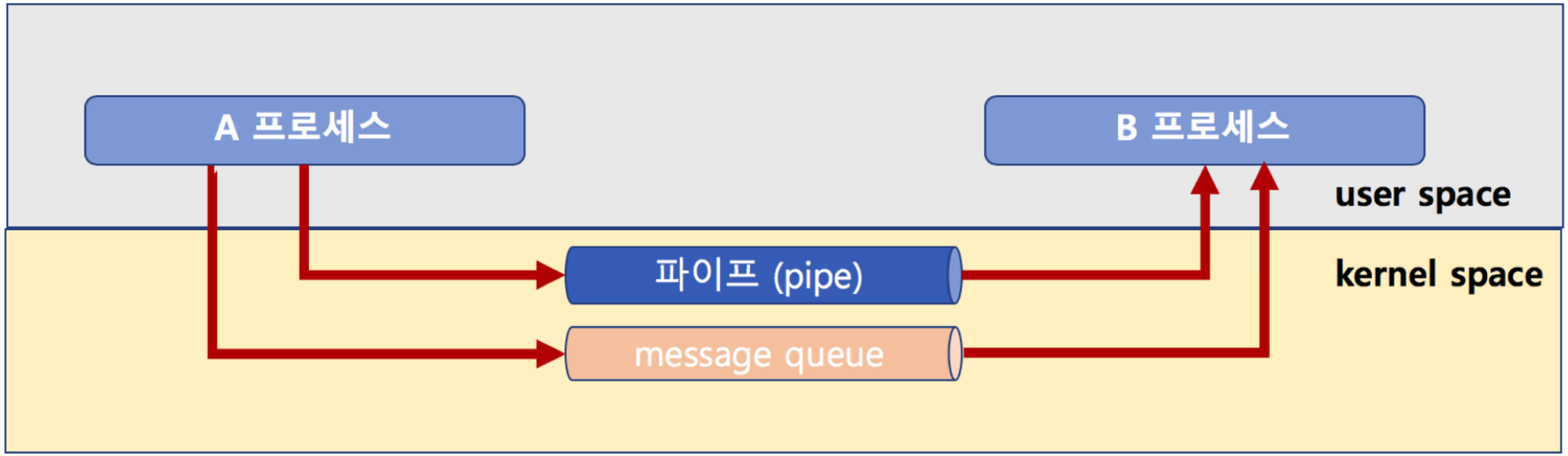
- message queue는 부모/자식이 아니라, 어느 프로세스간에라도 데이터 송수신이 가능
- 먼저 넣은 데이터가, 먼저 읽혀진다.

pipe VS message queue

- 부모/자식 프로세스간 only or not
- 단방향만 가능 or 양방향 가능

IPC 기법과 커널 모드

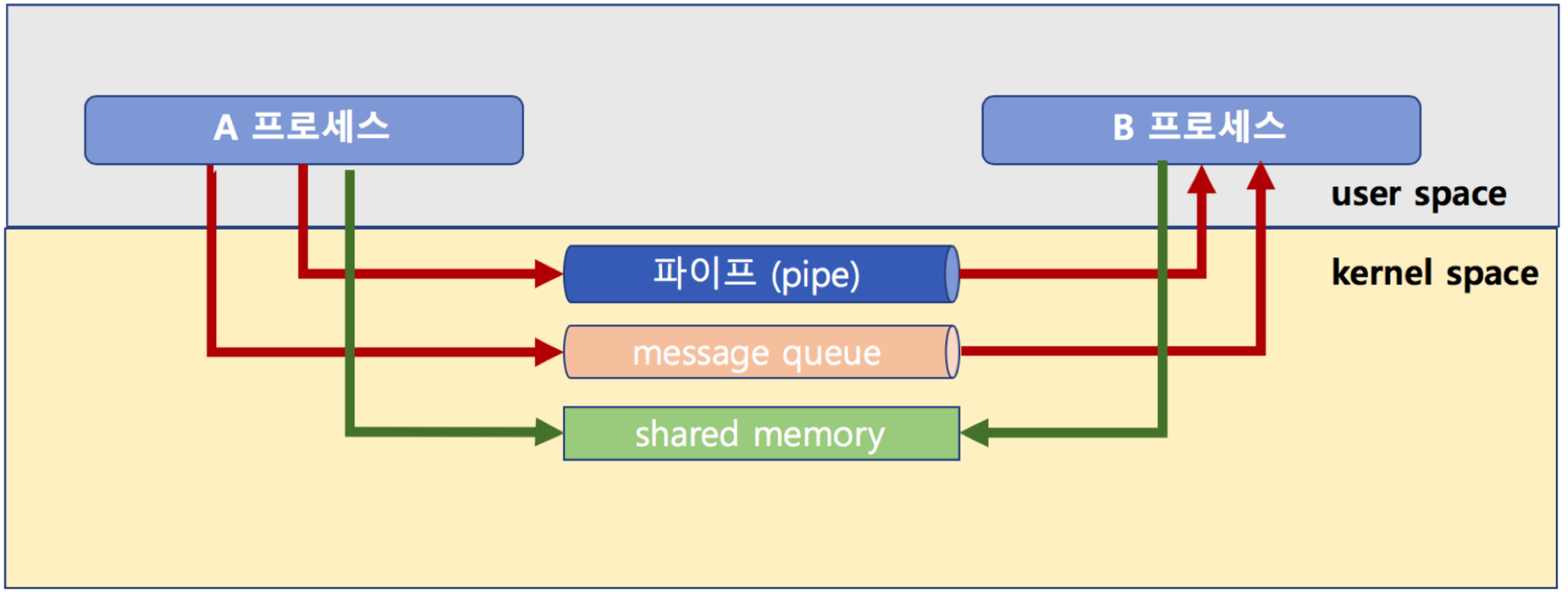
> pipe, message queue 는 모두 kernel 공간의 메모리를 사용합니다.



> 메모리 공간도 kernel/user 로 구분됩니다. 이 부분은 가상 메모리와 함께 다음에 이해해보겠습니다.

공유 메모리 (shared memory)

- 노골적으로 kernel space에 메모리 공간을 만들고, 해당 공간을 변수처럼 쓰는 방식
- message queue 처럼 FIFO 방식이 아니라, 해당 메모리 주소를 마치 변수처럼 접근하는 방식
- 공유메모리 key를 가지고, 여러 프로세스가 접근 가능



공유 메모리 코드 예제

1. 공유 메모리 생성 및 공유 메모리 주소 얻기

```
shmid = shmget((key_t)1234, SIZE, IPC_CREAT|0666))
shmaddr = shmat(shmid, (void *)0, 0)
```

2. 공유 메모리에 쓰기

```
strcpy((char *)shmaddr, "Linux Programming")
```

3. 공유 메모리에서 읽기

```
printf("%s\n", (char *)shmaddr)
```

정리

1. 주요 IPC 기법
    - pipe
    - message queue
    - shared memory
2. 모두 커널 공간을 활용해서 프로세스간 데이터 공유

> 이상은 기본적인 IPC

Fast campus

프로스세간 커뮤니케이션

- 각 IPC 기법 개념 이해하기 (signal과 socket)

이준희 강사 (dream@fun-coding.org)

프로세스간 커뮤니케이션

> IPC 기법이지만, 이외에도 많이 사용되는 두 가지 기술

- 많이 사용하는 두 가지!
  - 시그널 (signal)
  - 소켓 (socket)

> 간단하게만 짚고 넘어가기

시그널 (signal)

- 유닉스에서 30년 이상 사용된 전통적인 기법
- 커널 또는 프로세스에서 다른 프로세스에 어떤 이벤트가 발생되었는지를 알려주는 기법
- 프로세스 관련 코드에 관련 시그널 핸들러를 등록해서, 해당 시그널 처리 실행
    i. 시그널 무시
    ii. 시그널 블록(블록을 푸는 순간, 프로세스에 해당 시그널 전달)
    iii. 등록된 시그널 핸들러로 특정 동작 수행
    iv. 등록된 시그널 핸들러가 없다면, 커널에서 기본 동작 수행

주요 시그널

> 주요 시그널: 기본 동작

- SIGKILL: 프로세스를 죽여라(슈퍼관리자가 사용하는 시그널로, 프로세스는 어떤 경우든 죽도록 되어 있음)
- SIGALARM: 알람을 발생한다
- SIGSTP: 프로세스를 멈춰라(Ctrl + z)
- SIGCONT: 멈춰진 프로세스를 실행해라
- SIGINT: 프로세스에 인터럽트를 보내서 프로세스를 죽여라(Ctrl + c)
- SIGSEGV: 프로세스가 다른 메모리영역을 침범했다.

> 시그널 종류: kill -l

시그널 관련 코드 예제

- 시그널 핸들러 등록 및 핸들러 구현

```
static void signal_handler (int signo) {
        printf("Catch SIGINT!\n");
        exit (EXIT_SUCCESS);
}

int main (void) {
        if (signal (SIGINT, signal_handler) == SIG_ERR) {
            printf("Can't catch SIGINT!\n");
            exit (EXIT_FAILURE);
        }

        for (;;)
                pause();
        return 0;
}
```

시그널 관련 코드 예제

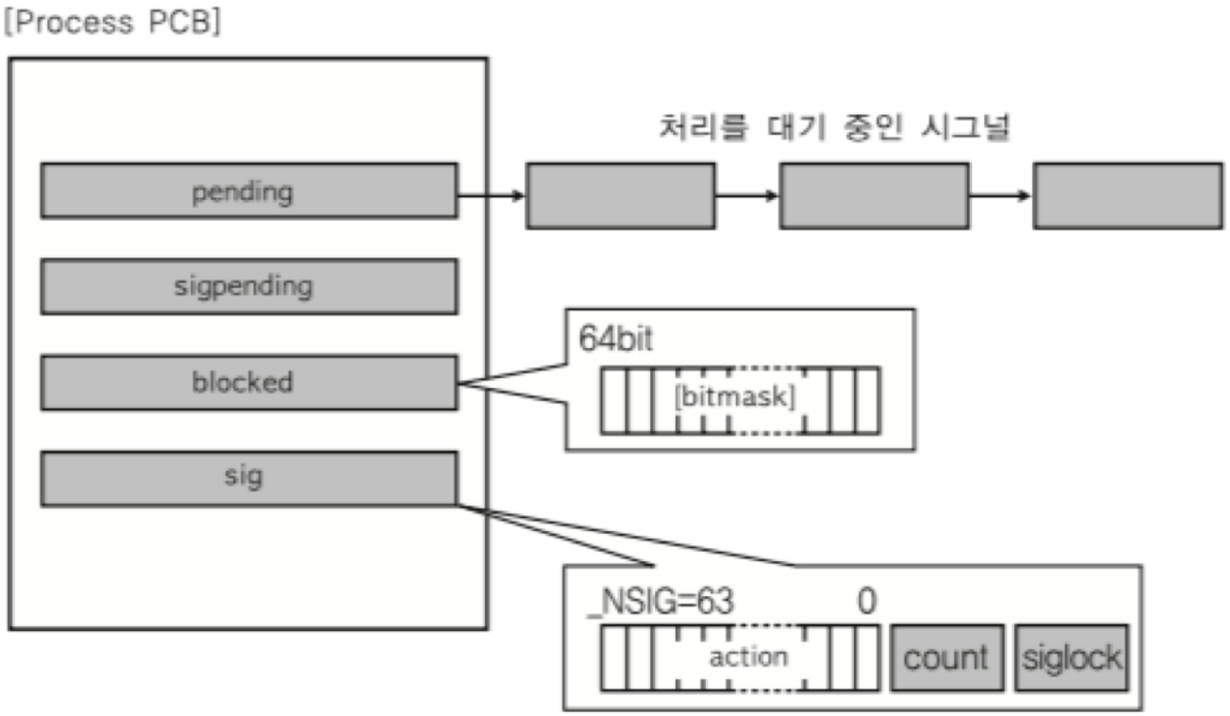
- 시그널 핸들러 무시

```
int main (void) {
        if (signal (SIGINT, SIG_IGN) == SIG_ERR) {
            printf("Can't catch SIGINT!\n");
            exit (EXIT_FAILURE);
        }

        for (;;)
                pause();
        return 0;
}
```

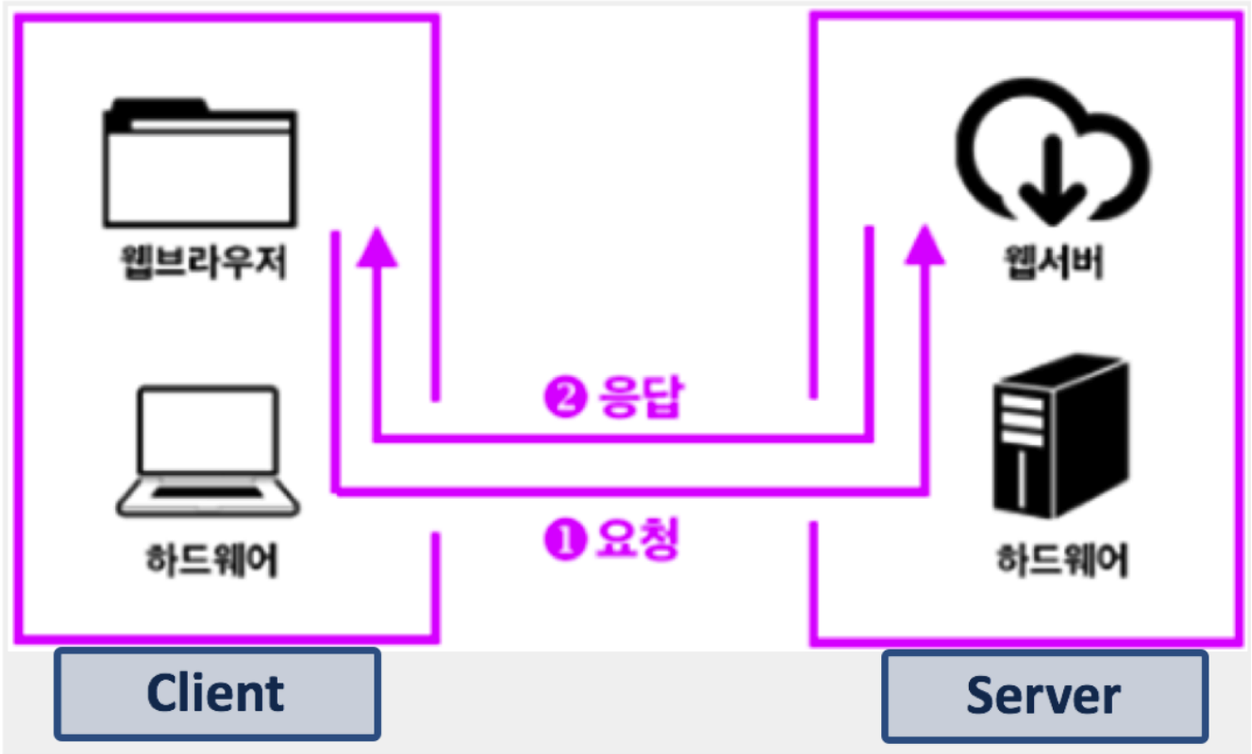
시그널과 프로세스

- PCB에 해당 프로세스가 블록 또는 처리해야하는 시그널 관련 정보 관리



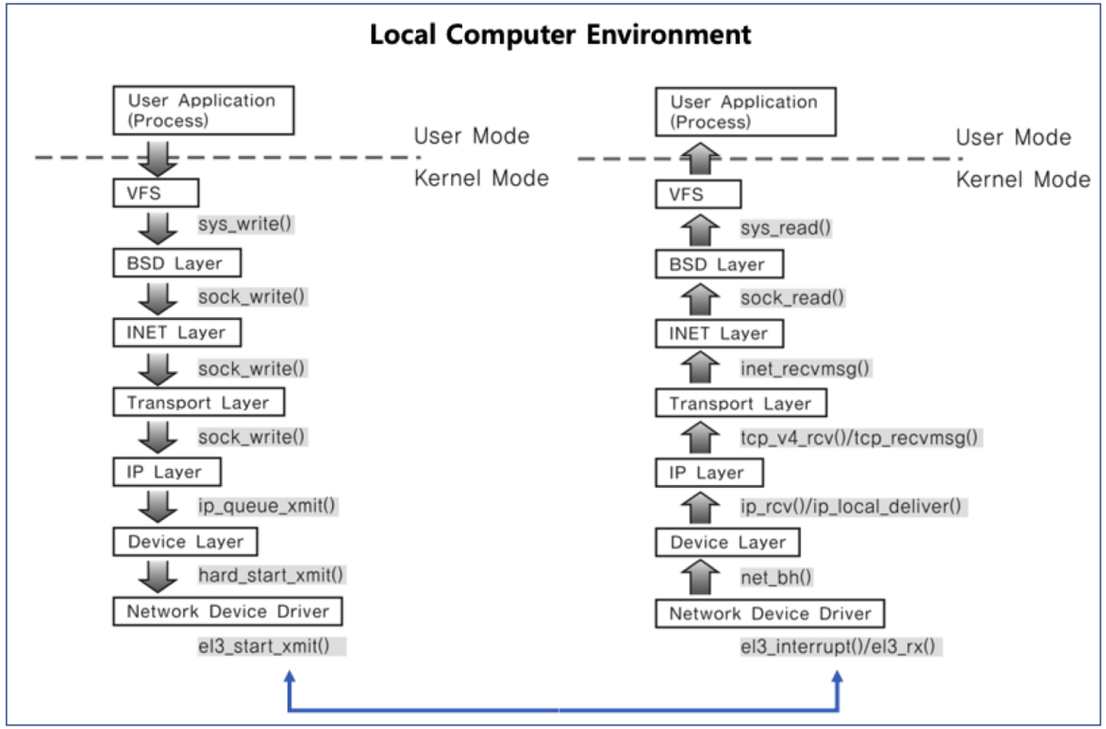
소켓(socket)

- 소켓은 네트워크 통신을 위한 기술
- 기본적으로는 클라이언트와 서버등 두 개의 다른 컴퓨터간의 네트워크 기반 통신을 위한 기술



소켓(socket)과 IPC

- 소켓을 하나의 컴퓨터 안에서, 두 개의 프로세스간에 통신 기법으로 사용 가능



정리

- 다양한 IPC 기법을 활용해서, 프로세스간 통신이 가능하다.
- IPC 기법 이외에도 사용할 수 있는 다음 두 가지 기술 개념을 알아둡니다.
  - 시그널(signal)
  - 소켓

> 실제 관련 코드에 대해서는 시스템 프로그래밍을 통해 실습 과정을 함께 진행해보겠습니다.