Network Programming - 02

afewhee@gmail.com

- Socket Mode
- I/O 모델
- Select I/O

- 소켓 모드(socket mode)
  - 소켓 함수 호출시 동작 방식에 따라 블로킹(blocking)과 넌블로킹(nonblocking) 소켓으로 구분

- 블로킹 소켓
  - 소켓 함수 호출 시 조건이 만족되지 않으면 함수는 리턴하지 않고 해당 스레드는 대기 상태(wait state)가 됨

소켓 함수	리턴 조건
accept()	클라이언트가 접속했을 때
send(), sendto()	송신 버퍼에 데이터를 모두 복사했을 때
recv(), recvfrom()	수신 버퍼에 도착한 데이터가 있을 때

- 넌블로킹 소켓
  - 소켓 함수 호출시 조건이 만족되지 않더라도 함수가 리턴하므로 해당 스레드는 계속 진행 가능
  - socket() 함수는 기본적으로 블로킹 소켓을 생성하므로 ioctlsocket() 함수를 호출하여 넌블로킹 소켓으로 전환

Ex)

```
// 블로킹 소켓 생성
SOCKET listen_sock = socket(AF_INET, SOCK_STREAM, 0);
...
// 넌블로킹 소켓으로 전환
u_long on = 1;
retval = ioctlsocket(listen_sock, FIONBIO, &on);

if(retval == SOCKET_ERROR)
    error Process;
```

- 넌블로킹 소켓과 소켓 함수
  - 넌블로킹 소켓에 대해 소켓 함수를 호출했을 때 조건이 만족되지 않아 작업을 완료할 수 없으면 소켓 함수는 오류를 리턴 ➔ 반드시 오류는 아님
  - WSAGetLastError() 함수를 호출하여 오류인지 확인
  - 대부분 WSAEWOULDBLOCK이 값이며되며, 이는 조건이 만족되지 않음을 나타내므로 나중에 다시 소켓 함수를 호출해야 함

- 넌-블로킹 소켓의 특징
  - 장점:
    - 소켓 함수 호출 시 블록되지 않으므로 다른 작업 진행 가능
    - 멀티스레드를 사용하지 않고도 여러 개의 소켓 입출력 처리 가능
  - 단점:
    - 소켓 함수를 호출할 때마다 WSAEWOULDBLOCK 등 오류 코드를 확인하고, 다시 해당 함수를 호출해야 하므로 프로그램 구조가 복잡해짐
    - 블로킹 소켓을 사용한 경우보다 CPU 사용률이 높음

- 반복 서버
  - 접속한 여러 클라이언트를 하나씩 차례대로 처리
  - 장점: 하나의 스레드로 클라이언트를 처리하므로 시스템 자원 소모가 적음
  - 단점: 서버와 클라이언트의 통신 시간이 길어지면 다른 클라이언트의 대기 시간이 길어짐

- 병행 서버
  - 접속한 여러 클라이언트를 병렬적으로 처리
  - 장점: 서버와 클라이언트의 통신 시간이 길어지더라도 다른 클라이언트의 통신에 영향을 주지 않음
  - 단점: 멀티프로세스 또는 멀티스레드를 이용하여 구현하므로 시스템 자원 소모가 큼

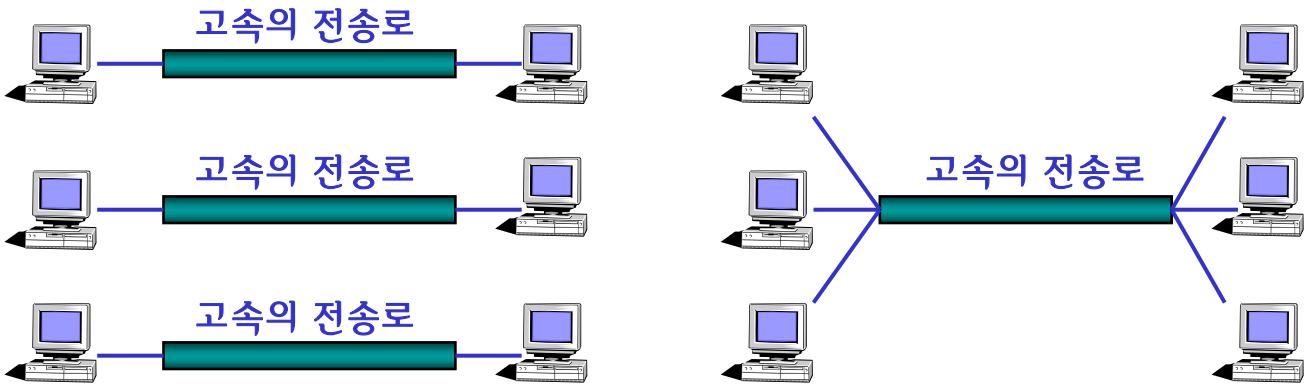
- 이상적인 서버의 특징
  - 모든 클라이언트 접속이 성공
  - 서버는 각 클라이언트의 서비스 요청에 최대한 빠르게 반응하며, 고속으로 데이터를 전송
  - 시스템 자원 사용량을 최소화

- 이상적인 소켓 입출력 모델
  - 소켓 함수 호출 시 블로킹 최소화
  - 입출력 작업을 다른 작업과 병행
  - 스레드 개수를 최소화
  - 유저 모드와 커널 모드 전환 횟수와 데이터 복사를 최소화

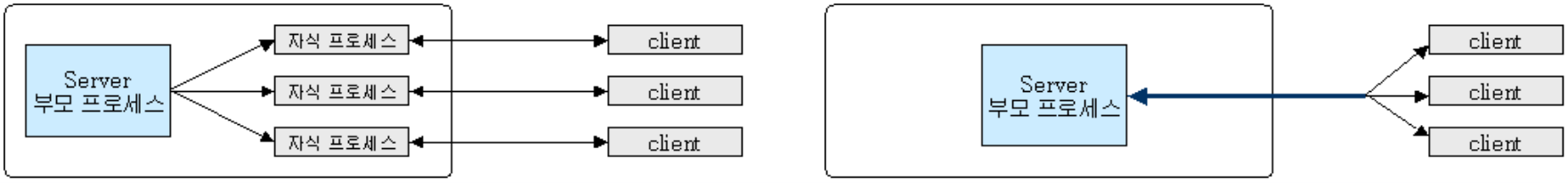
● 윈도우 운영 체제와 소켓 입출력 모델 지원

소켓 입출력 모델	지원 운영체제		
	윈도우 CE	윈도우(클라이언트 버전)	윈도우(서버 버전)
Select	CE 1.0 이상	윈도우 95 이상	윈도우 NT 이상
WSAAsyncSelect	x	윈도우 95 이상	윈도우 NT 이상
WSAEventSelect	CE .NET 4.0 이상	윈도우 95 이상	윈도우 NT 3.51 이상
Overlapped	CE .NET 4.0 이상	윈도우 95 이상	윈도우 NT 3.51 이상
Completion Port (IOCP)	x	윈도우 NT 3.5 이상 (윈도우 95/98/Me 제외)	

- Multiplexing
  - 데이터를 전송하는데 있어서, 물리적인 장치의 효율성을 높이기 위해 최소한의 물리적인 요소만 사용하여 최대한의 정보를 주고 받기 위해 개발된 방법
    MUX (Multiplexer)



- I/O Multiplexing
  - - 통신 객체 사이의 입/출력을 담당하는 프로세스를 하나로 처리



- 멀티 프로세스 기반 서버
  - 클라이언트와 서버간의 송수신 데이터 용량이 큰 경우.
  - 송수신이 연속적으로 발생 하는 경우에 적합.

- 멀티플렉싱 기반 서버
  - 클라이언트와 서버간의 송수신 데이터 용량이 작은 경우.
  - 송수신이 연속적이지 않은 경우에 적합
  - 멀티 프로세스 기반의 서버에 비해 많은 수의 클라이언트 처리에 적합.

- Select 모델
  - 멀티플렉싱을 구현한 모델
  - select() 함수로 구현
  - 하나의 스레드로 소켓 모드(블로킹, 넌-블로킹)에 관계없이 여러 개의 소켓을 처리

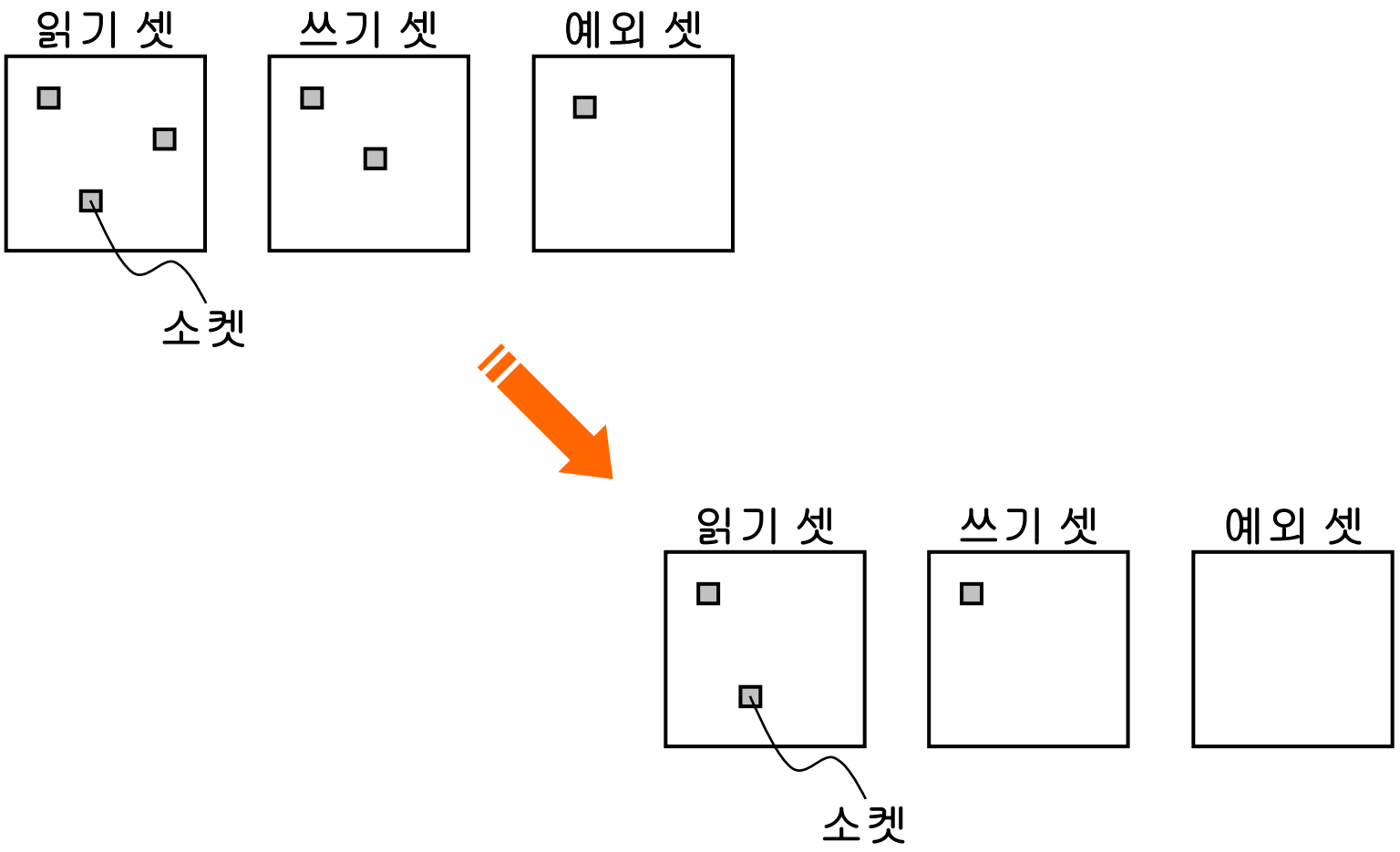
- 핵심 원리
  - 소켓 함수를 호출해야 할 시점을 알려줌으로써 함수 호출 시 항상 성공하도록 함
    - 블로킹 소켓: 소켓 함수 호출 시 조건이 만족되지 않아 블로킹 되는 상황을 방지
    - 넌블로킹 소켓: 소켓 함수 호출 시 조건이 만족되지 않아 다시 호출해야 하는 상황을 방지

- select 함수

```
int select (
    int nfds,
    fd_set* readfds,
    fd_set* writefds,
    fd_set* exceptfds,
    const struct timeval* timeout
) ;
```

  - 성공: 조건을 만족하는 소켓의 개수
  - 0: 타임아웃
  - 실패: SOCKET_ERROR

- 동작 원리



● 소켓 셋 (각각 읽기, 쓰기, 예외 셋)

함수 호출 시점	✓ 클라이언트가 접속했으므로 accept() 함수를 호출할 수 있다. ✓ 데이터를 받았으므로 recv(), recvfrom() 등의 함수를 호출할 수 있다. ✓ 연결이 종료되었으므로 recv(), recvfrom() 등의 함수를 호출할 수 있다. 이때 리턴값은 0 또는 SOCKET_ERROR가 된다.

함수 호출 시점	✓송신 버퍼가 충분하므로 send(), sendto() 등의 함수를 호출하여 데이터를 보낼 수 있다.
함수 호출 결과	✓ 넌블로킹 소켓을 사용한 connect() 함수 호출이 성공하였다.

함수 호출 시점	✓ OOB(Out-Of-Band) 데이터가 도착했으므로 recv(), recvfrom() 등의 함수를 호출하여 OOB 데이터를 받을 수 있다.
함수 호출 결과	✓ 넌블로킹 소켓을 사용한 connect() 함수 호출이 실패하였다.

- select() 함수를 이용한 소켓 입/출력 절차
  - ※ select 함수를 사용할 수 있도록 인수를 설정하는 과정
    - 1. 검사 해야 할 소켓의 범위 지정
      - 실제로는 검사해야 하는 소켓의 개수를 인수로 전달
      - 가장 큰 소켓 값에 1을 더해서 인수로 전달
    - 2. 타임 아웃을 설정: struct timeval 이용
    - 3. 소켓 셋을 비움(초기화)
    - 4. 소켓 셋에 소켓을 추가. 셋에 넣을 수 있는 소켓의 최대 개수는 FD_SETSIZE(=64)로 정의. ➔ 헤더 파일을 수정해서 크기 조정 가능
  - 5. select() 함수를 호출.
    - 블로킹: select() 함수는 조건이 만족되는 소켓이 있을 때까지 리턴 안 함
    - 넌-블로킹: 대부분 오류로 리턴하므로WSAGetLastError() 함수로 반드시 확인
  - 6. select() 함수가 리턴한 후 소켓 셋에 존재하는 모든 소켓에 대해 적절한 소켓 함수를 호출하여 처리
  - 7. 1~6을 반복

- FD_CLR(SOCKET s, fd_set *set)
  - 셋에서 소켓 s를 제거
- FD_ISSET(SOCKET s, fd_set *set)
  - 소켓 s가 셋에 들어 있으면 0이 아닌 값을 리턴. 그렇지 않으면 0을 리턴
- FD_SET(SOCKET s, fd_set *set)
  - 셋에 소켓 s를 넣음
- FD_ZERO(fd_set *set)
  - 셋을 비움