ㅋ

일일 업무 사항 정리

작성자	제품팀 이민성 인턴
업무 일시	20221212~20221230

세부 사항

1. 업무 내역 요약 정리

목표 내역	Done & Plan
	11주차 - 네트워크 프로그래밍 1. 소켓 프로그래밍 개요 - 소켓의 의미 - TCP/IP 소켓 통신이란 - TCP/IP란 - 소켓의 종류 - 소켓 통신 흐름 2. 다중 클라이언트 처리 - 개요 - 멀티프로세싱 활용 - 멀티스레딩 활용 - 멀티플렉싱 활용 3. 서버를 만들기 위한 절차

2. 내용 세부 (업무 세부 내역 정리 및 기타 사항 정리)

네트워크 프로그래밍 (소켓 프로그래밍)

1. 소켓 프로그래밍 개요

1. 소켓의 의미

* 사전적으로 구멍, 연결, 콘센트 등의 의미
* 프로그램이 네트워크에서 데이터를 송수신할 수 있도록, 네트워크 환경에 연결할 수 있게 만들어진 연결부
* 전원 소켓처럼 쓰고싶을 때 연결해서 사용한다는 의미 내포

2. TCP/IP 소켓 통신이란

* 전기 소켓이 전기를 공급받기 위해 정해진 규격에 맞게 만들어져야 하듯, 네트워크에 연결하기 위한 소켓 또한 정해진 규약, 즉, 통신의 위한 프로토콜에 맞게 만들어져야 함
* 보통 OSI 7 Layer(Open System Interconnection 7 Layer)의 네 번째 계층인 TCP(Transport Control Protocol) 상에서 동작하는 소켓을 주로 사용하는데, 이를 TCP 소켓 또는 TCP/IP 소켓이라고 부름

3. TCP/IP란

* TCP/IP는 패킷 통신 방식의 인터넷 프로토콜인 IP와, 전송 조절 프로토콜인 TCP로 이루어져 있음
* IP는 패킷 전달 여부를 보증하지 않고, 패킷을 보낸 순서와 받는 순서가 다를 수 있음
* TCP는 IP위에서 동작하는 프로토콜로, 데이터의 전달을 보증하고 보낸 순서대로 받게 해줌

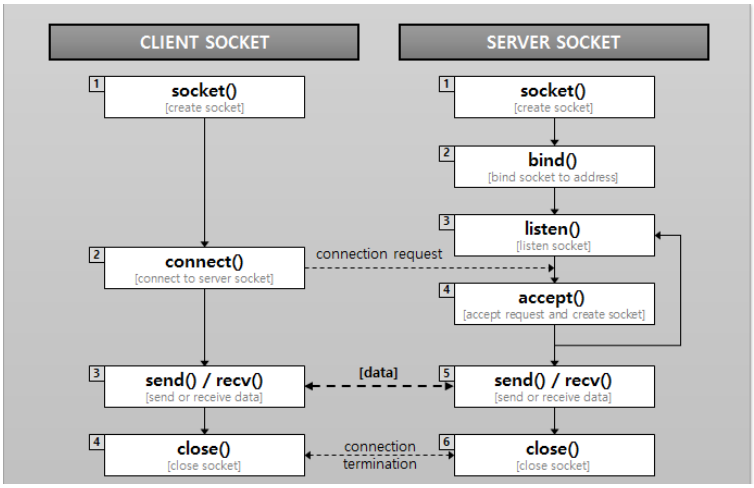
4. 소켓의 종류

* 통신 연결 요청을 보내는지 또는 요청을 받아들이는지에 따라 소켓의 역할이 나뉨
* 통신 연결 요청을 보내는 소켓을 클라이언트 소켓이라고 함
* 통신 연결 요청을 받아들이는 소켓을 서버 소켓이라고 함
* 두 소켓은 동일한 구조이고, 역할에 따라 처리되는 흐름이 다름

<주의>

* 소켓 연결이 완료된 다음 클라이언트 소켓과 서버 소켓이 직접 데이터를 주고받는다고 생각하면 안 됨
* 서버 소켓은 클라이언트 소켓의 연결 요청을 받아들이는 역할만 수행할 뿐, 직접적인 데이터 송수신은 서버 소켓의 연결 요청 수락의 결과로 만들어지는 새로운 소켓을 통해 처리됨

5. 소켓 통신 흐름



<서버 소켓>

1. 소켓을 생성함
2. 서버가 사용할 IP 주소와 포트 번호를, 생성한 소켓에 결합시킴
3. 클라이언트로부터 연결 요청이 수신되는지 주시함
4. 요청이 수신되면 요청을 받아들여 데이터 통신을 위한 새로운 소켓을 생성함
5. 연결 후 데이터를 송수신
6. 데이터 송수신이 완료되면, 소켓을 닫음

<클라이언트 소켓>

1. 소켓을 생성함
2. 서버 측에 연결을 요청함
3. 서버 소켓에서 연결이 받아들여지면 데이터를 송수신함
4. 모든 처리가 완료되면 소켓을 닫음

<주의 사항>

* 최종적으로 클라이언트 소켓과 연결이 만들어지는 소켓은 앞서 사용한 서버 소켓이 아니라, accept API 내부에서 새로 만들어지는 소켓임
* 실질적인 데이터 송수신은 accept API에서 생성된, 연결이 수립된 소켓을 통해 처리됨

2. 다중 클라이언트 처리

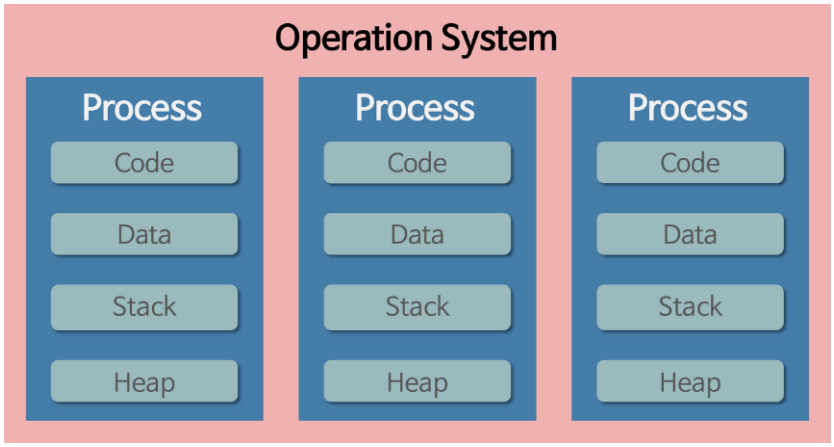
1. 개요

* 기본 서버-클라이언트 모델에서 서버는 한 번에 하나의 클라이언트만 처리함
* 다중 클라이언트를 처리하려면, 멀티프로세싱, 멀티스레딩, 멀티플렉싱 등을 활용해야 함

2. 멀티프로세싱 활용

[프로세스란]

* 프로세스의 의미는 Program in execution (실행 중인 프로그램)
* 프로세스는 각각 독립된 메모리 영역을 할당받음
* 기본적으로 프로세스당 최소 1 개의 스레드를 가지고 있음
* 각 프로세스는 별도의 주소 공간에서 실행되며, 한 프로세스는 다른 프로세스의 변수나 자료구조에 접근할 수 없음
* 한 프로세스가 다른 프로세스의 자원에 접근하려면 프로세스 간의 통신을 사용해야 함

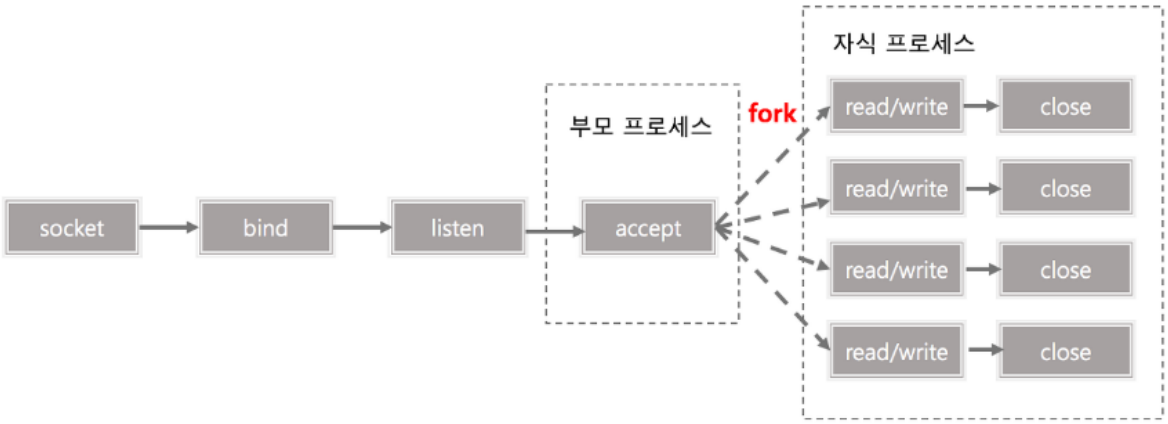


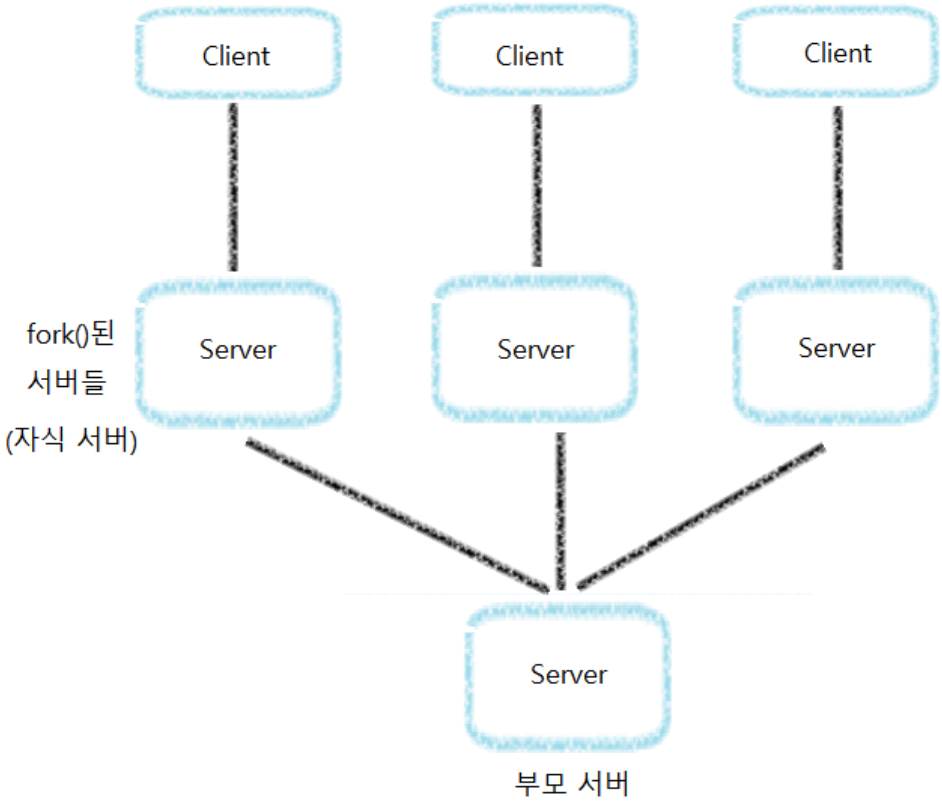
<프로세스 구성>

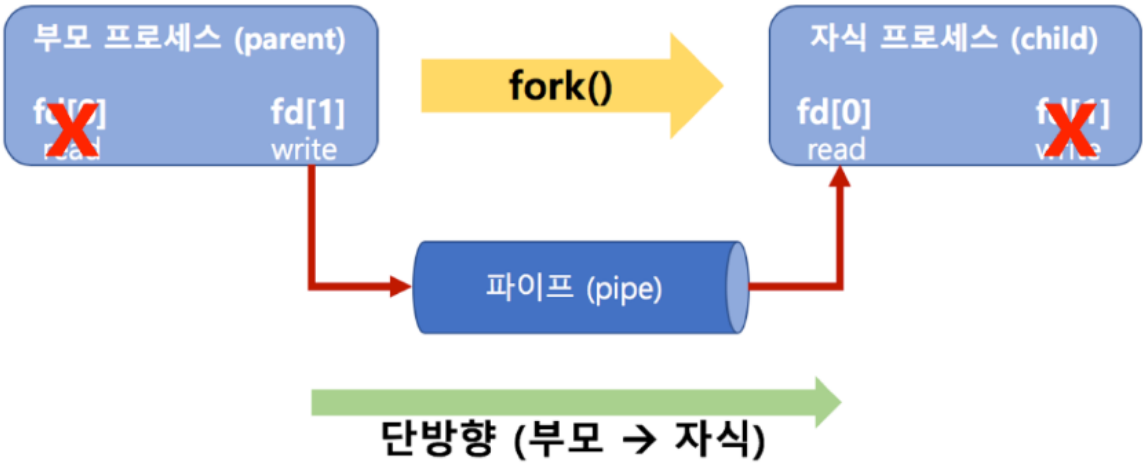
* stack: 지역변수 할당과 함수 호출 시 전달되는 인자값들을 저장하기 위한 공간
* heap: C의 malloc, calloc와 c++, java의 new를 통한 동적 할당을 위해 존재하는 공간
* data: 전역 변수나 static 변수의 할당을 위해 존재하는 공간
* code: 프로그램을 실행시키면 실행파일 내에 존재하는 명령어가 메모리 상에 올라가야 프로그램을 동작시킬 수 있음. 이 명령어들을 위해 존재하는 공간

<멀티프로세싱>

* 말 그대로, 프로세스를 여러 개 사용하여 클라이언트를 처리할 수 있음
* 프로세스 당 하나의 클라이언트를 처리함
* 흔히 알고 있는, fork()를 통해 멀티 프로세스를 만들 수 있음
* 부모 프로세스 1 개에 자식 프로세스 N개를 만들 수 있음
* 부모 프로세스와 자식 프로세스는 독립 프로세스로 각각 실행됨
* 아래 그림에서 fork 뜬 하나하나가 프로세스임



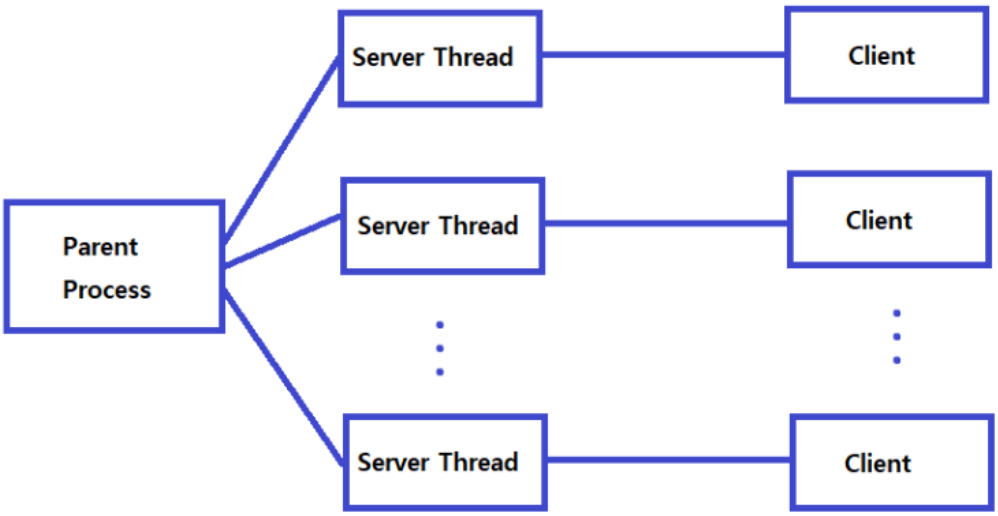


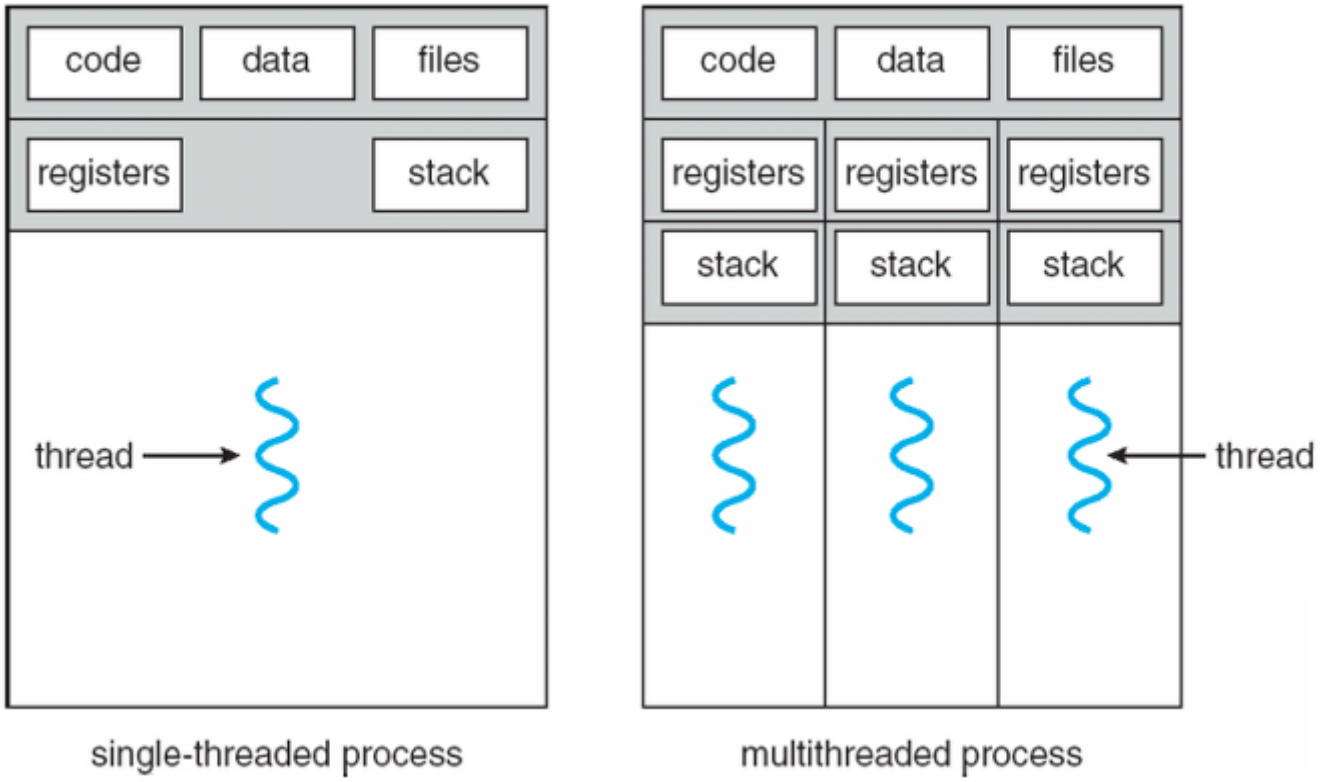


<장점>
* 독립된 구조이기 때문에 안정성이 높음
* 클라이언트와 서버 간의 송수신 데이터 용량이 큰 경우 적합
* 송수신이 쉬지 않고 연속적으로 발생하는 경우 적합

<단점>
* 프로세스가 많이 생성될수록 메모리 사용량이 증가하고 프로세스 스케줄링 횟수도 많아져서 프로그램 성능이 떨어지게 됨
* 프로세스들 사이에 데이터를 공유할 수 없음. 프로세스 간 데이터를 공유하려면 운영체제의 도움을 받아 프로세스 간 통신을 해야함. 이로 인해 프로그램 구현이 복잡해질 수 있음

3. 멀티스레딩 활용
* 여러 클라이언트를 처리하는 간단한 방법은, 서버에 연결된 모든 새 클라이언트에 대해 새 스레드를 생성하는 것임
* 스레드는 프로세스보다 작은 단위이며 프로세스 안에서 논리적으로 동작하는 하나의 작업단위
* 두 개 이상의 스레드를 가지고 있는 프로세스를 멀티스레드 프로세스라고 함
* 외부에서는 스레드를 전체가 하나의 프로세스처럼 취급됨





<장점>

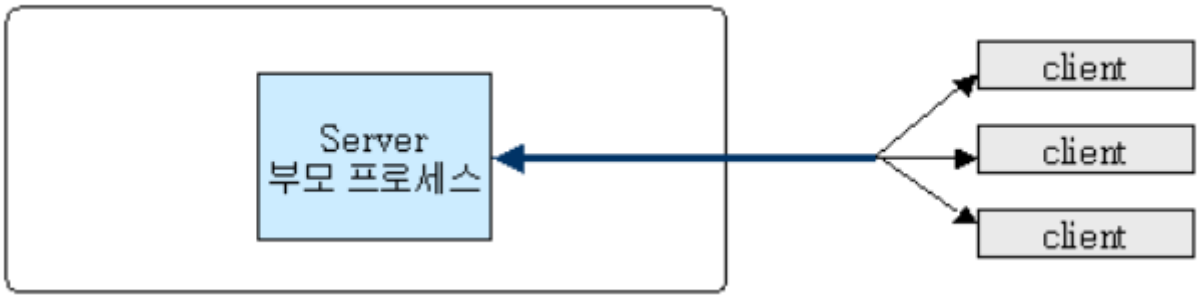
* IPC(프로세스 간 통신)에 비해 스레드 간 통신 방법이 훨씬 간단함
* 스레드는 stack 영역을 제외한 모든 메모리를 공유하여 시스템 자원 소모가 줄어 듦
* 멀티프로세싱보다는 Context Switching에 대한 오버헤드가 줄어 듦 (멀티프로세스를 통해 PCB를 Context Switching 하는 것보다 멀티스레드를 통해 TCB를 Context Switching 하는 비용이 더 적다고 알려져 있음)

<단점>

* 스레드는 코딩, 디버깅이 어렵고 때로는 예측할 수 없는 결과가 발생함. 여러 개의 스레드를 이용하는 경우, 미묘한 시간차나 잘못된 변수를 공유함으로써 오류 발생 가능
* 많은 수의 클라이언트에 대해 확장 불가능
* 교착상태가 발생할 수 있음
* 서버에 접속한 각 클라이언트 별로 read() 스레드를 할당할 경우, 데이터가 오지 않으면 계속 블로킹 상태가 되기 때문에 자원이 낭비될 수 있음

4. 멀티플렉싱 활용

* 멀티플렉싱이란, 하나의 전송로를 여러 사용자가 동시에 사용해서 효율성을 극대화하는 것
* I/O 멀티플렉싱이란, 클라이언트와 입/출력하는 프로세스를 하나로 묶어버리는 형식 (프로세스가 고속의 전송로에 해당)

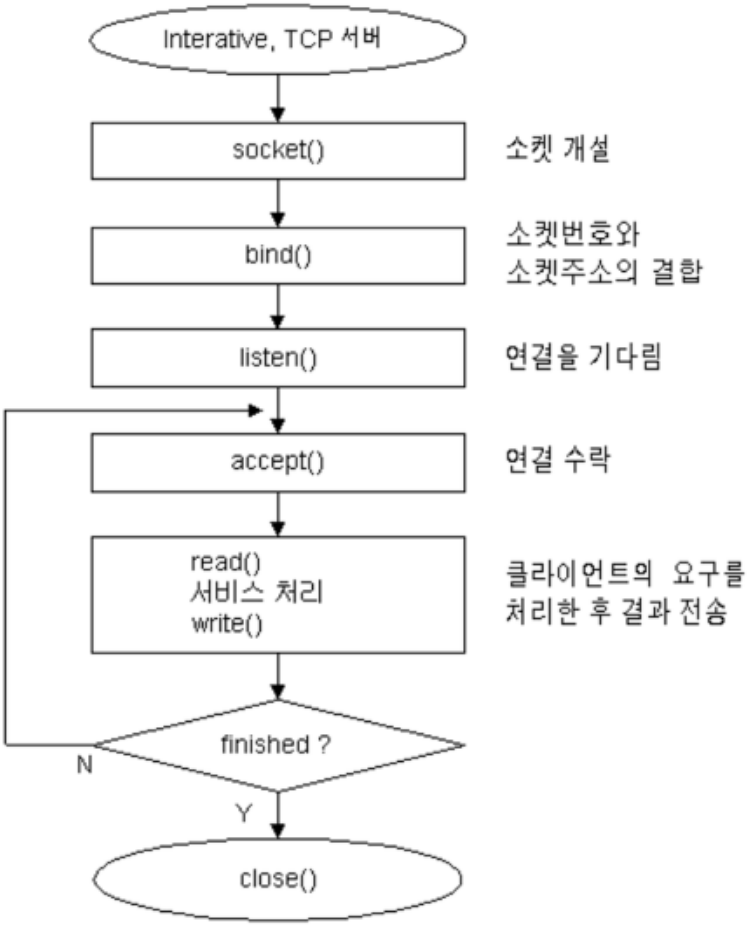


* 하나의 스레드를 이용하여 여러 클라이언트와 동시에 메시지를 주고받을 수 있음
* 서버에서 여러 명의 클라이언트와 동시에 메시지를 주고받기 위한 멀티스레드는 각 스레드 당 메모리와 CPU의 스택을 필요로 하기 때문에 서버의 성능이 저하될 수 있음
* 이러한 단점을 보완하기 위해 하나의 스레드만 생성하여, 여러 클라이언트와 통신할 수 있는 IO 멀티플렉싱이 등장
* 지금까지의 흐름을 정리하자면, 여러 개의 프로세스(멀티프로세싱) 하나의 프로세스 & 여러 개의 스레드(멀티스테딩) 단일 스레드 (멀티플렉싱)
* 멀티플렉싱 설계 시 코드가 복잡해짐. 논블로킹을 위해 핸들러를 만들어야 하며, 콜백 개념도 이해해야 함
* 하나의 스레드에 작업량이 많아지는 것을 고려해 Thread Pool을 만들어 Task들을 분산처리 해랴 함
* 이러한 것들을 쉽게 처리할 수 있는 함수가 있음

<장점>

* 스레드 하나가 여러 개의 클라이언트를 관리할 수 있어서 굉장히 효울적임
* 클라이언트와 서버 간 송수인 데이터의 용량이 적은 경우 적합
* 송수인이 연속적이지 않은 경우에 적합
* 멀티프로세스에 비해 많은 수의 클라이언트 처리에 적합

3. 서버를 만들기 위한 절차



1. socket() 소켓 생성

통신하기위한 통신 개찰구인 소켓을 생성해준다.

2. bind()

내가 만드려는 서버의 ip 주소와 포트를 소켓에 할당하는 역할을 수행한다

3. listen()

다른 컴퓨터에서 노크를 두드려도 응답해줄 수 있도록 대기상태로 만든다

4. accept()

만약 어떤 클라이언트로부터 연결 요청이 왔으면 수락해주는 함수이다

5. read() write()

두 컴퓨터가 연결되었으니 필요한 데이터를 전송하고 받는다

6. close()

끝났으면 연결을 끊어준다

1. socket() 소켓 생성

소켓을 생성하는 함수이다.

```
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

인자값:

int domain:

어떤 영역에서 통신할 것인지에 대한 영역을 지정한다.

올 수 있는 값: AF_UNIX, AF_INET, AF_INET6 등

AF_UNIX는 프로세스끼리 통신할 때, AF_INET은 IPv4, AF_INET6 는 IPv6 를 의미한다.

int type:

어떤 서비스 타입의 소켓을 생성할 것인지 적는 것이다.

SOCK_STREAM(TCP), SOCK_DGRAM(UDP), SOCK_RAW(Raw 방식 TCP나 UDP를 거치지 않고 바로 IP계층 사용시)

int protocol:

소켓에서 사용할 프로토콜

IPPROTO_TCP: TCP방식

IPPROTO_UDP: UDP방식

0: type에서 미리 정해진 경우

리턴값:

소켓을 가리키는 소켓 디스크립터를 반환한다

-1 소켓 생성 실패

0 이상의 값 : 소켓 디스크립터

* 디스크립터: 리눅스 혹은 유닉스 계열의 시스템에서 프로세스(process)가 파일(file)을 다룰 때 사용하는 개념

```
//TCP연결지향형이고 ipv4 도메인을 위한 소켓을 생성
serv_sock=socket(PF_INET, SOCK_STREAM, 0);
if(serv_sock == -1)
    printf("socket error\n");
```

2. bind()

```
1 #include <sys/socket.h>
2
3 int bind(int sockfd,struct sockaddr *myaddr, socklen_t addrlen);
```

소켓이랑 서버의 정보를 묶어주는 함수이다. 다른 외부의 컴퓨터가 서버에 연결하려고 요청을 해서 IP주소를 기반으로 찾았겠지만 통신을 위해서 소켓디스립터 번호를 알아야하는데 IP주소를 안다고 소켓디스립터 번호를 알 수 있는 것이 아니라서 두 개를 묶어주는 작업을 하는 것이다.

인자값:

int sockfd: fd가 파일디스크립터의 약자로, 1 번 함수 리턴값으로 받은 소켓 디스크립터를 여기 넣어주면 된다.

struct sockaddr *myaddr: 서버의 ip주소를 넣어준다

scoklen_t addrlen: 주소 길이를 넣어준다

리턴값: 성공시 0, 실패시 -1

```
// 소켓과 서버 주소를 바인딩
if(bind(serv_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr))==-1)
        printf("bind error");
```

3. listen()

어떤 컴퓨터로부터 요청이 와도 수락할 수 있게 대기상태에 들어가는 함수이다.

```
1 #include <sys/socket.h>
2
3 int listen(int sockfd, int backlog);
```

인자값:

int sockfd: 2 번 bind 함수의 첫 인자와 같다. 소켓 디스크립터를 여기 넣어준다.

itn backlog: 연결 대기열의 크기를 지정한다. 소켓이 대기하는 연결 대기열을 생성한다. 네트워크 상태와 서비스 종류에 따라서 달라진다.

리턴값: 성공시 0, 실패시 -1

4. accept()

```
1 #include <sys/socket.h>
2
3 int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

서버 소켓에다가 클라이언트를 연결하는 함수이다. 그래서 2, 3 번째 주소 관련 인자에는 클라이언트 정보가 들어간다.

인자값:

int sockfd: 2 번 bind 함수의 첫 인자와 같다. 서버 소켓 디스크립터를 여기 넣어준다.

struct sockaddr *addr: 클라이언트 주소 정보를 담고 있는 구조체

socklen_t *addrlen: 2 번째 인자 값의 길이