

2025년 1학기 시스템프로그래밍실습 7주차

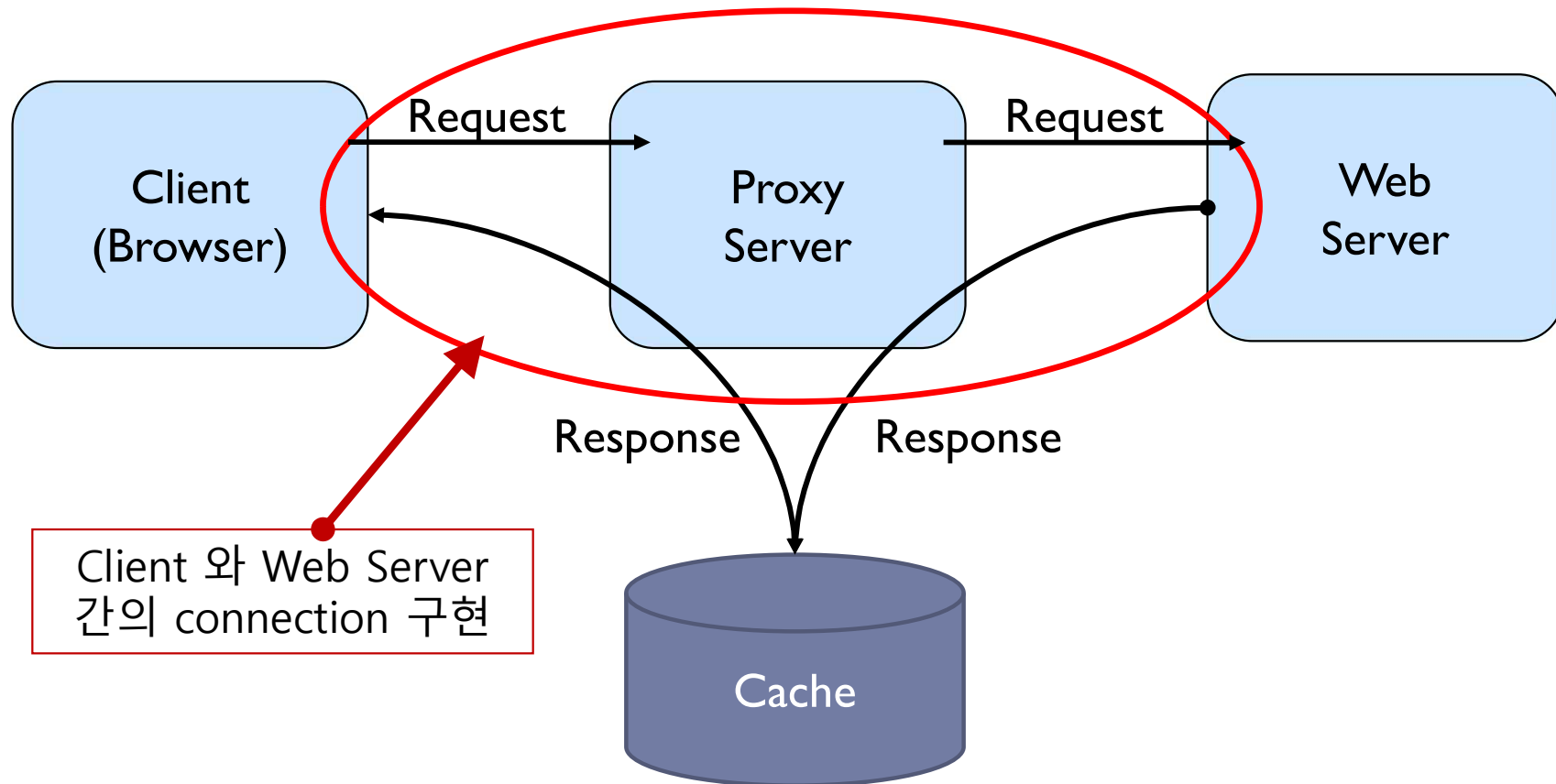
Socket Programming

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

2nd Assignment's Descriptions

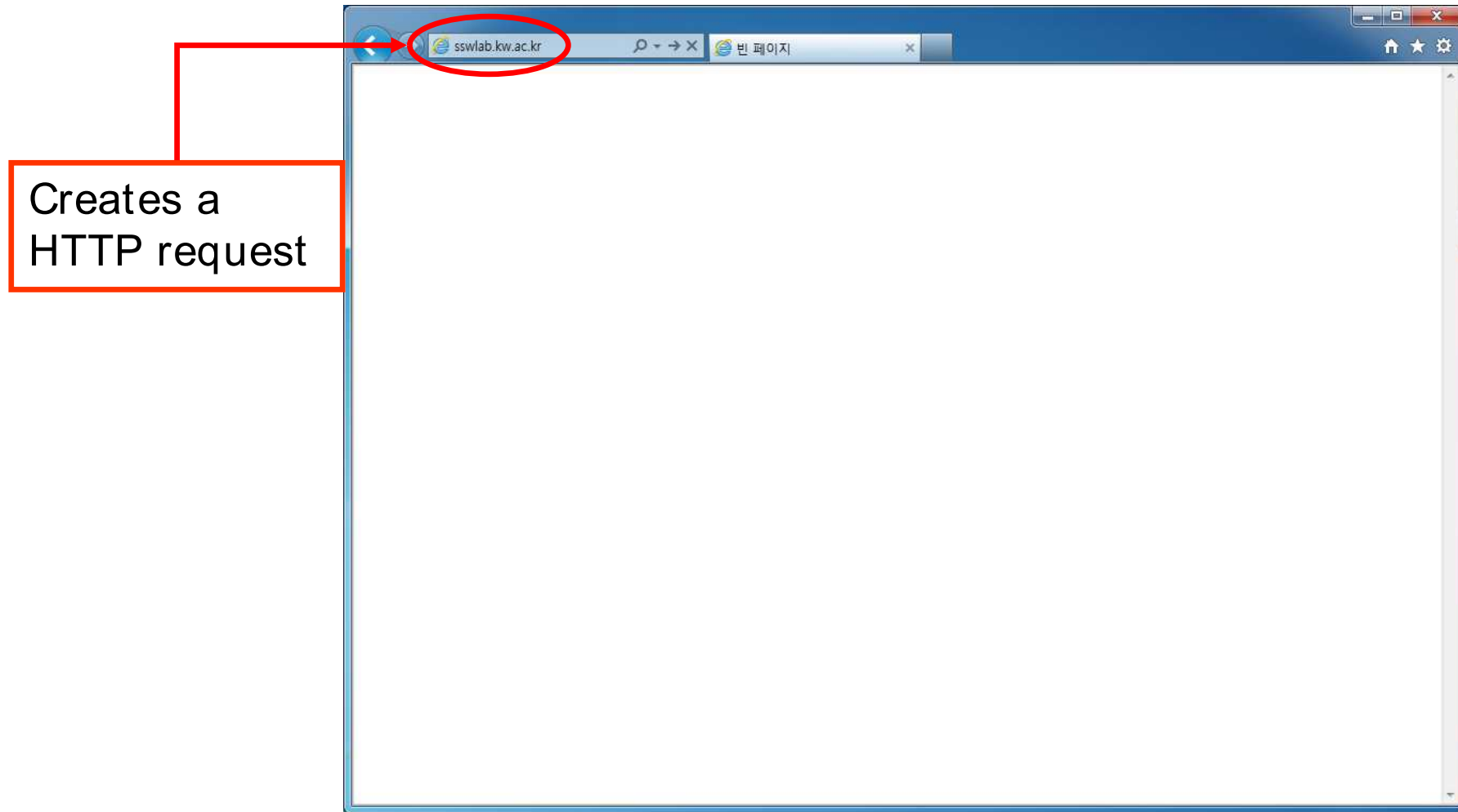
- **Assignment 2-1**
 - Implement server/client
- **Assignment 2-2**
 - HTTP request handling in proxy server
- **Assignment 2-3**
 - Forward HTTP request to web server and signal handling
- **Assignment 2-4**
 - Add cache and log to proxy server

Overview



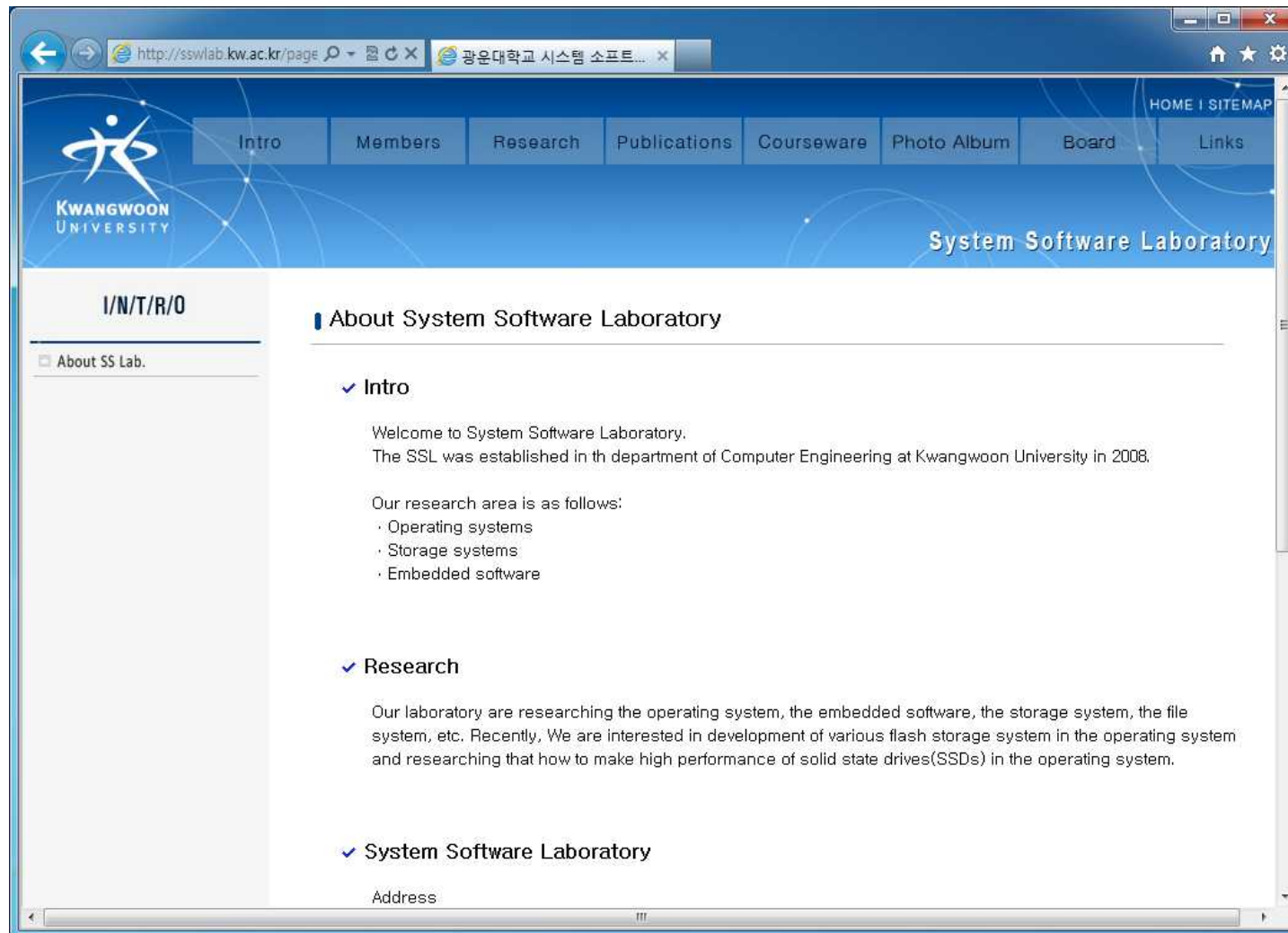
Proxy Server's Input

- A HTTP request from a client

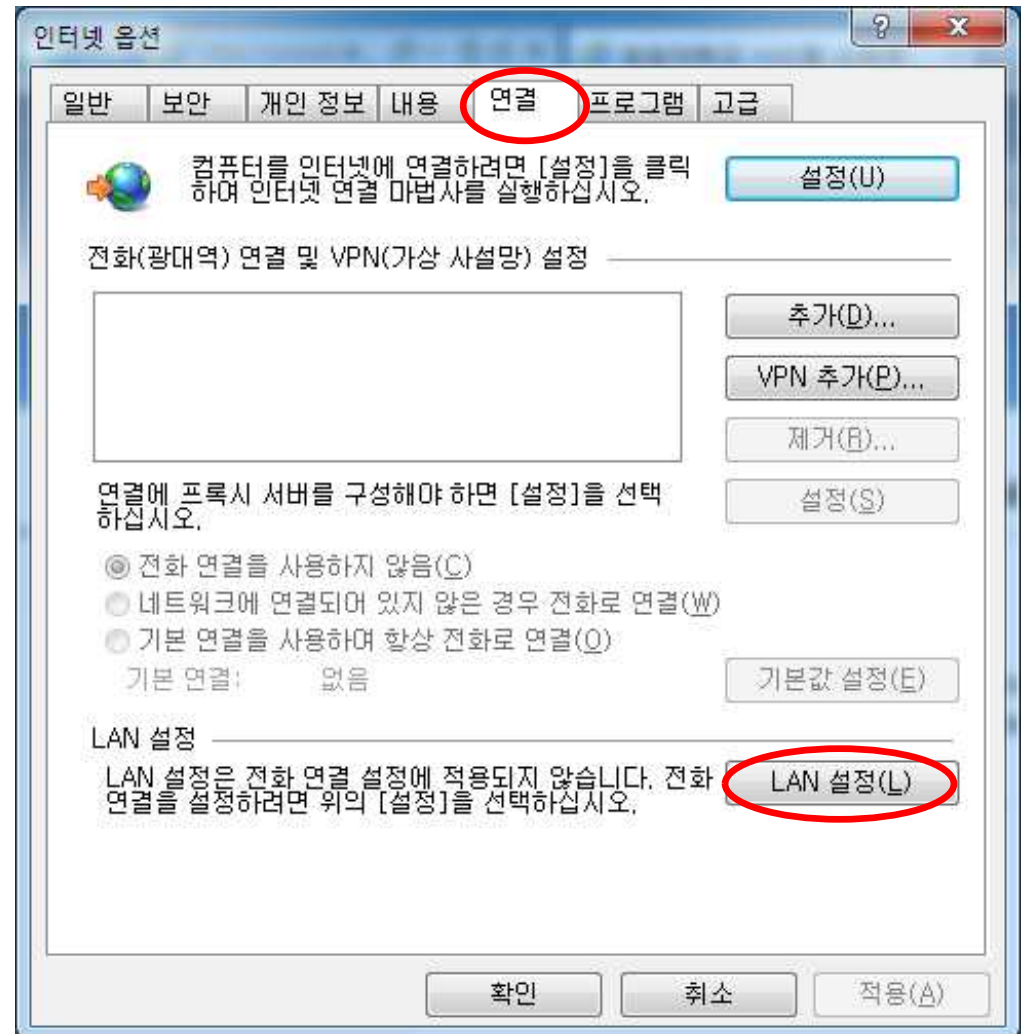
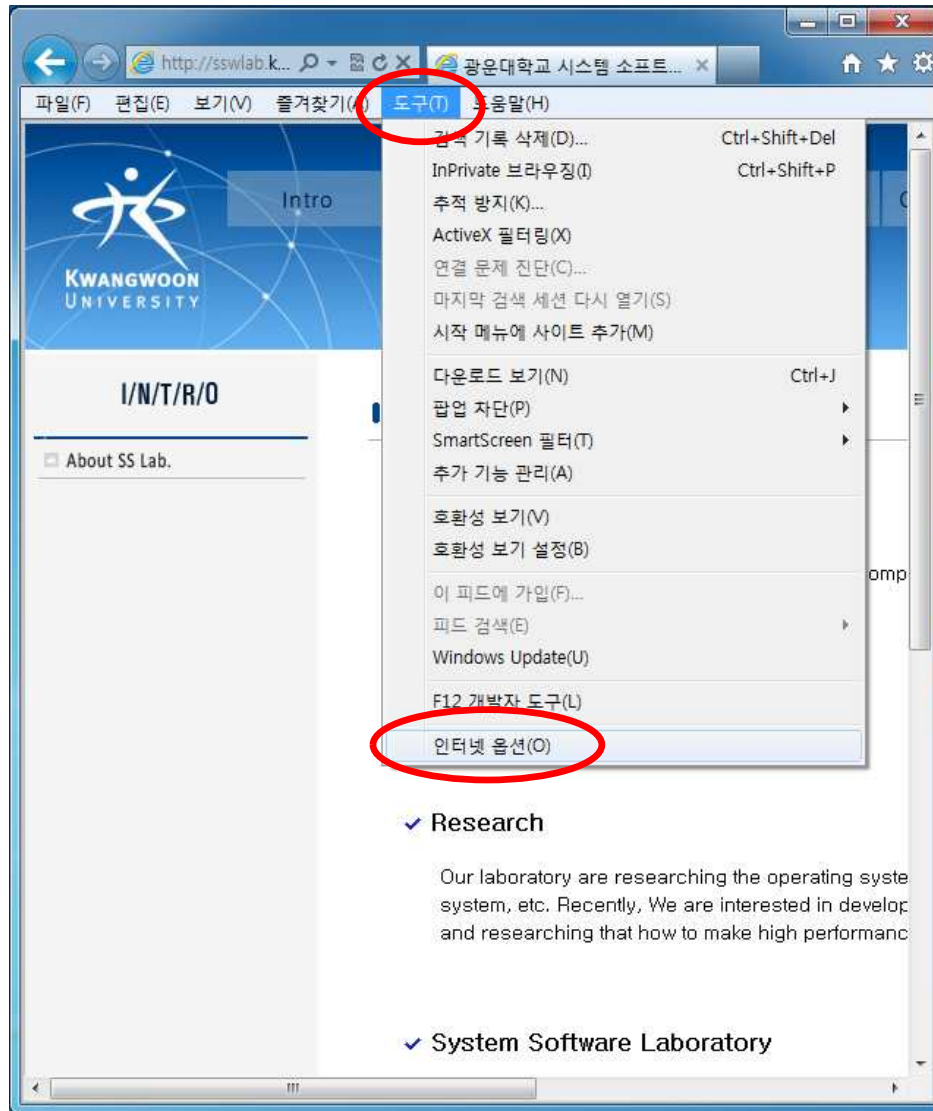


Proxy Server's Output

- A HTTP response from the Web server through a proxy server



Web Browser의 Proxy 설정 (1/2)



Web Browser의 Proxy 설정 (2/2)

LAN 설정

자동 구성
자동 구성은 수동 설정보다 우선합니다. 수동 설정을 사용하려면 자동 구성을 사용하지 마십시오.

☒ 자동으로 설정 검색(A)
☐ 자동 구성 스크립트 사용(S)
주소(R):

프록시 서버
☒ 사용자 LAN에 프록시 서버 사용(이 설정은 전화 연결이나 VPN 연결에는 적용되지 않음)(X)
주소(E): 223.194.46.163 포트(T): 고급(C)
☐ 로컬 주소에 프록시 서버 사용 안 함(B)

확인 취소

프록시 설정

서버

종류	사용할 프록시 주소	포트
HTTP(H):	223.194.46.163	
Secure(S):	223.194.46.163	
FTP(F):	223.194.46.163	
Socks(C):		

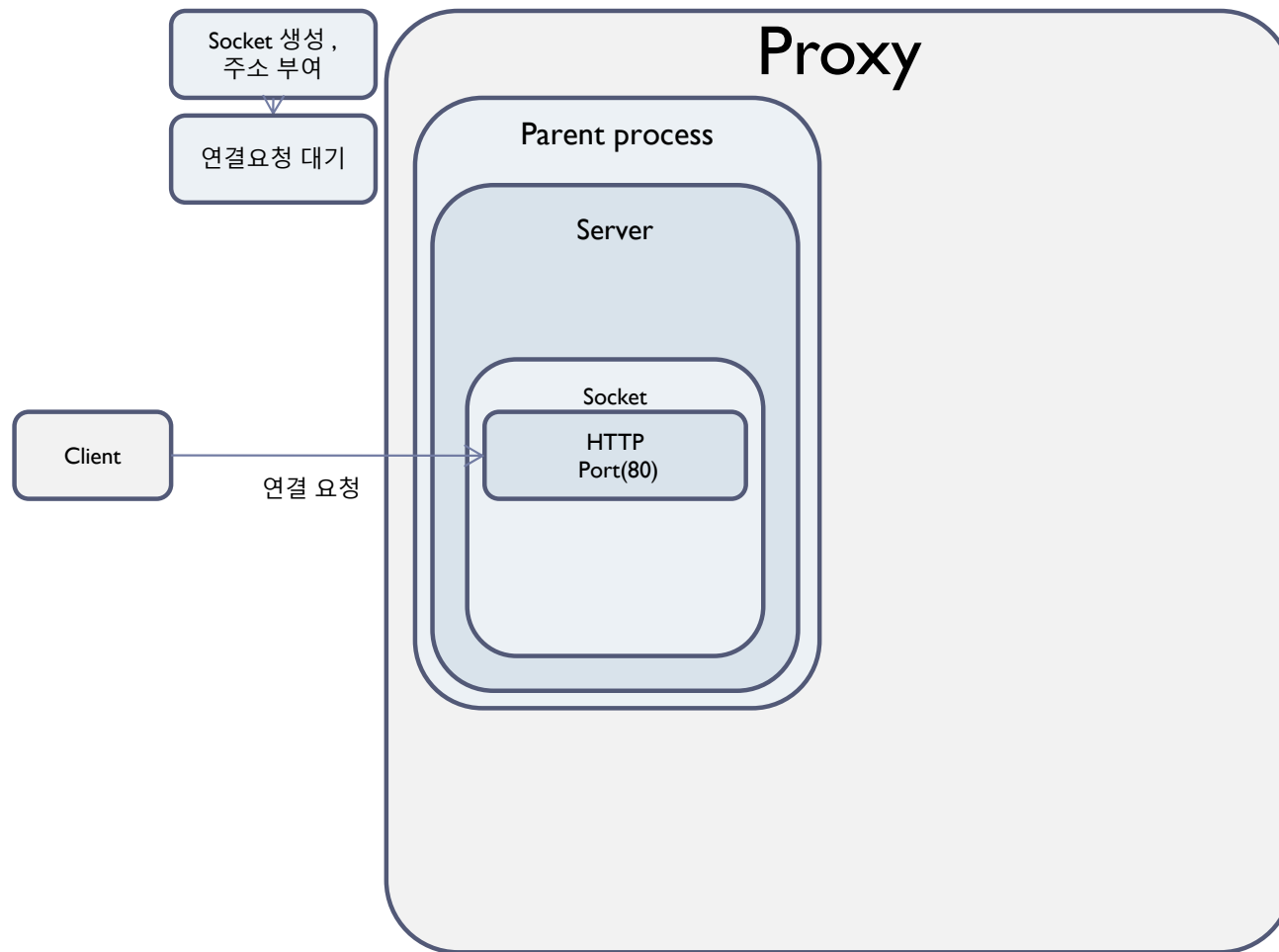
☒ 모든 프로토콜에 같은 프록시 서버 사용(U)

예외
다음으로 시작하는 주소에는 프록시 서버 사용 안 함(N):
127.0.0.1:16105;127.0.0.1:16106;127.0.0.1:21300;127.0.0.1:16107;127.0.0.1:16108;localhost
항목이 여러 개인 경우 세미콜론(;)을 사용하여 구분

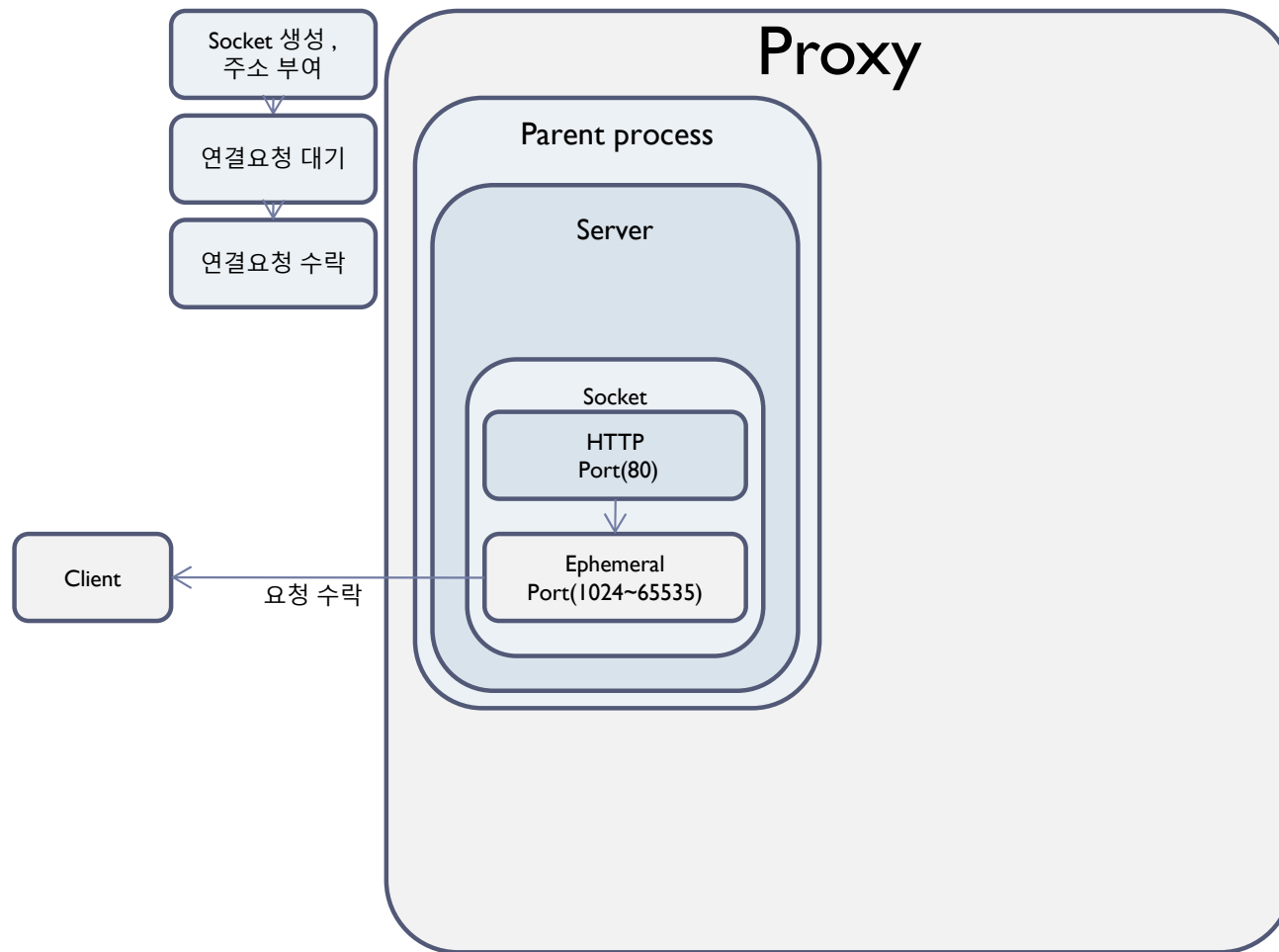
확인 취소

각 학생들이 할당 받은 포트 번호 사용

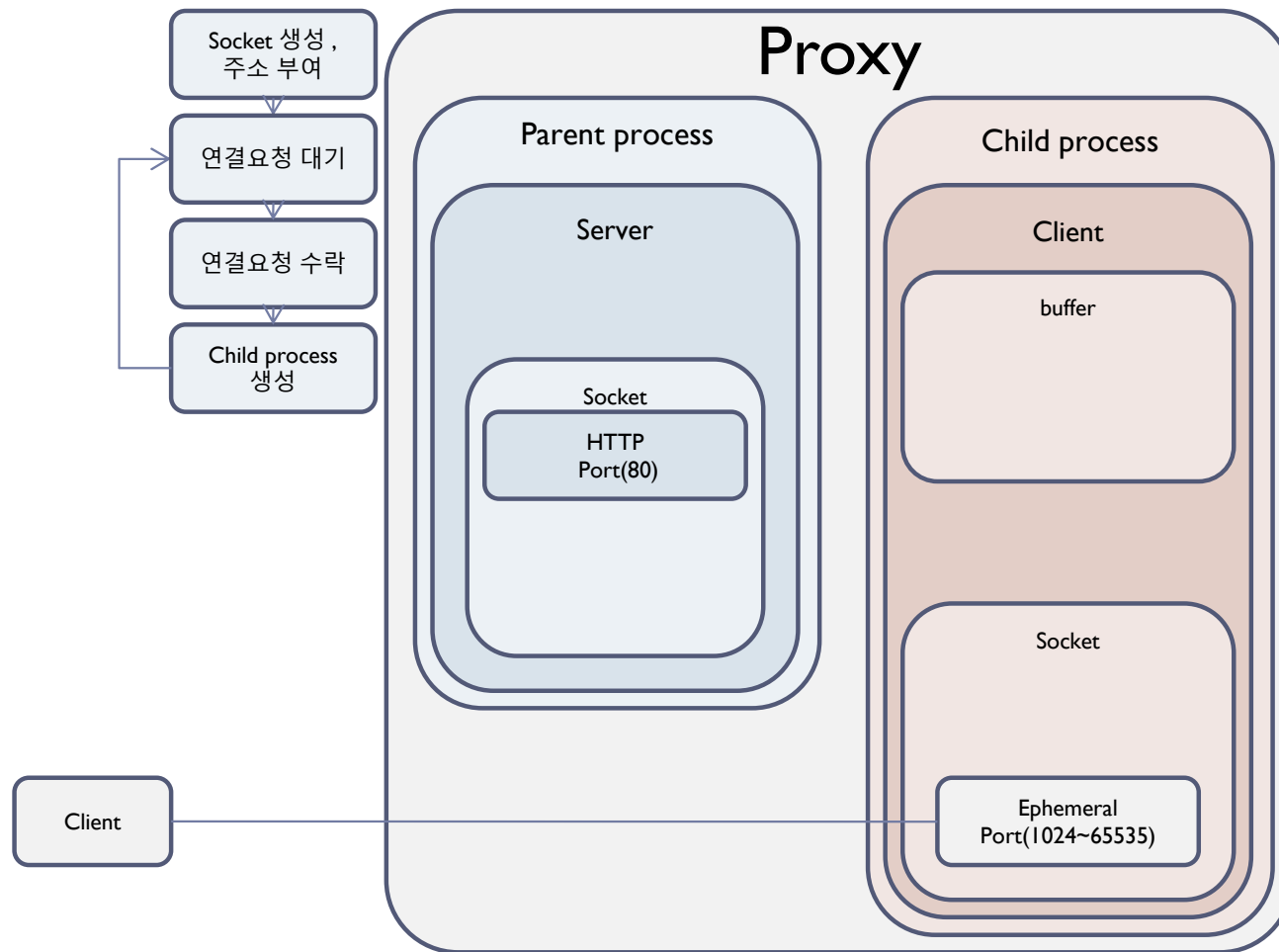
Proxy Connection (1/9)



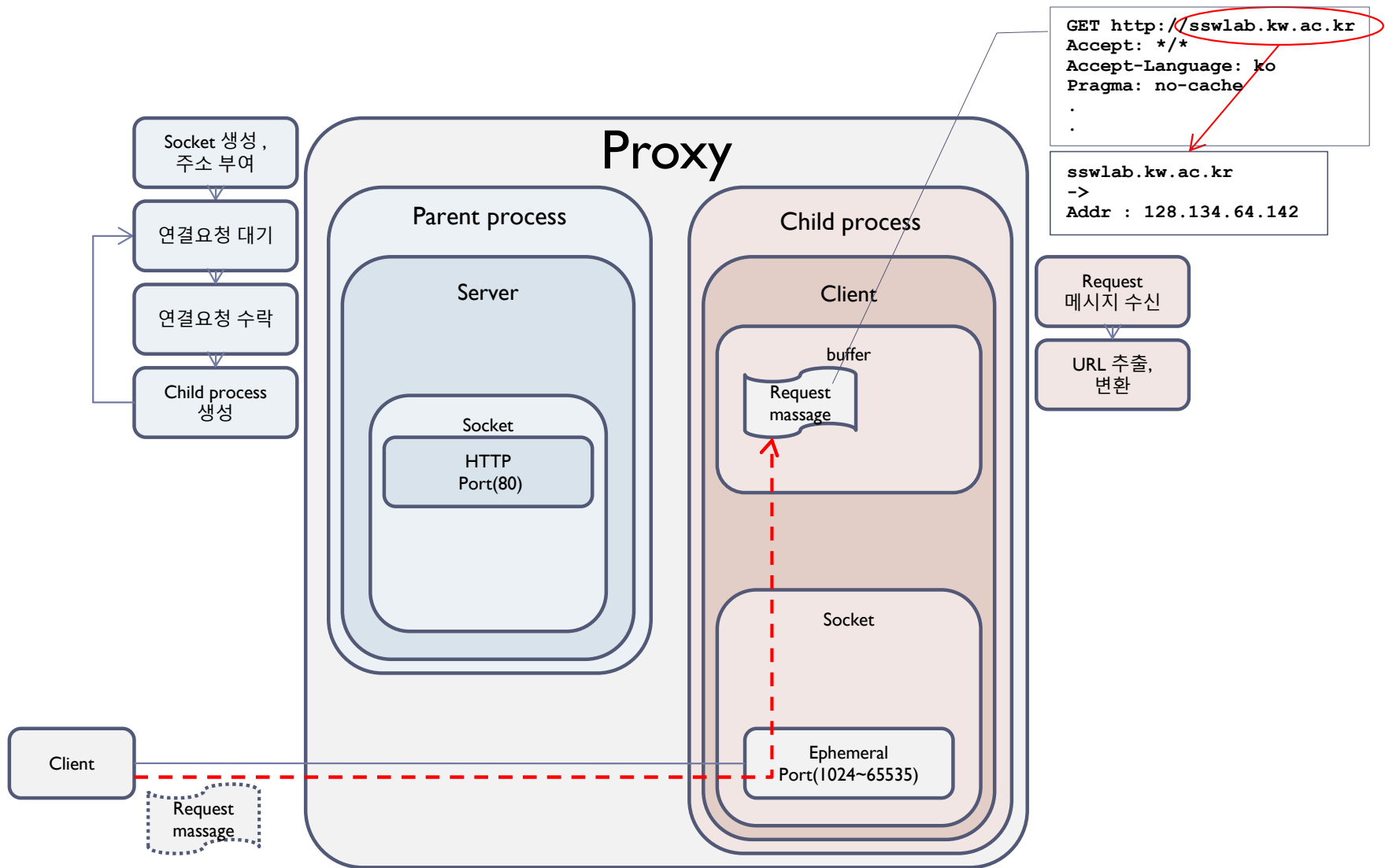
Proxy Connection (2/9)



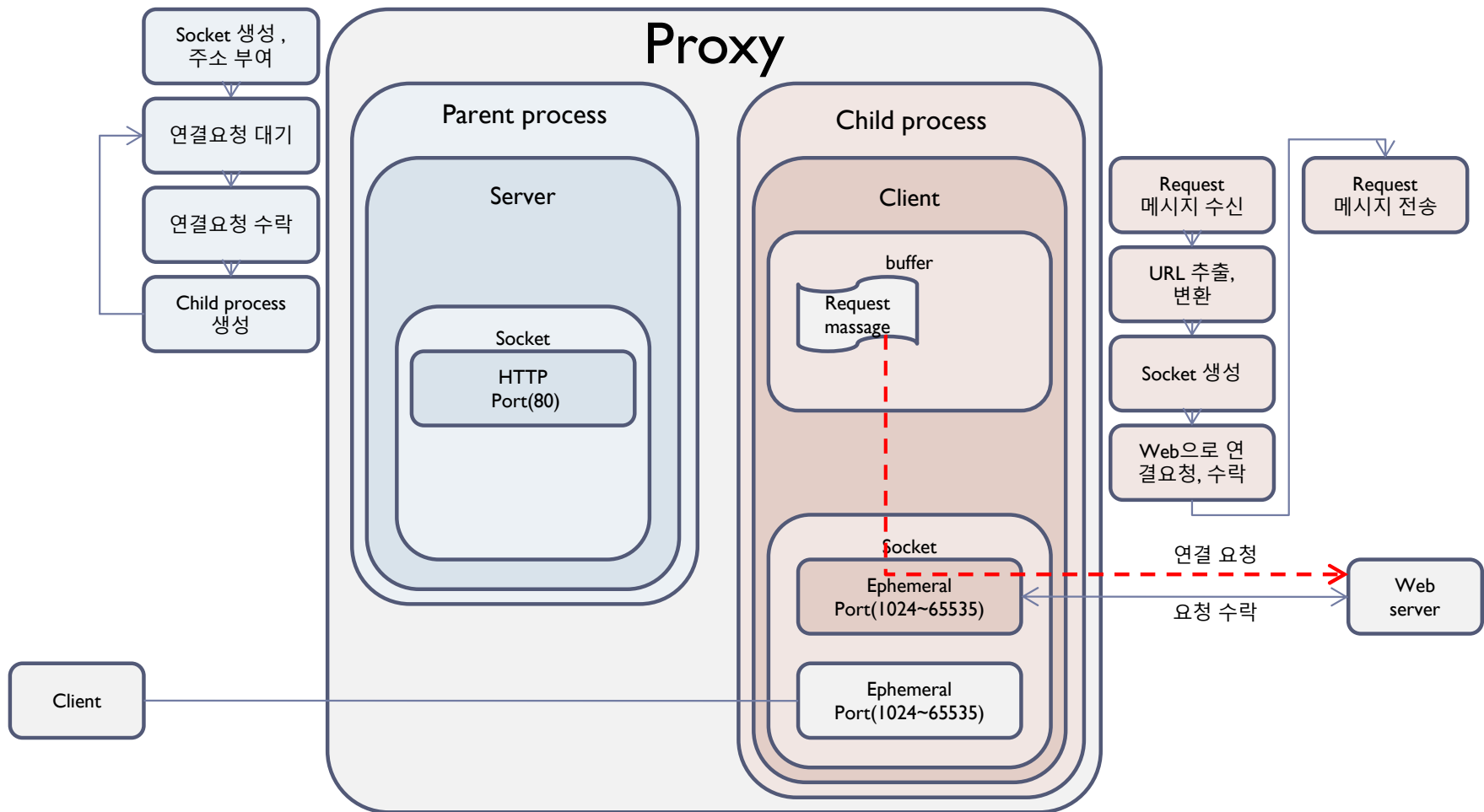
Proxy Connection (3/9)



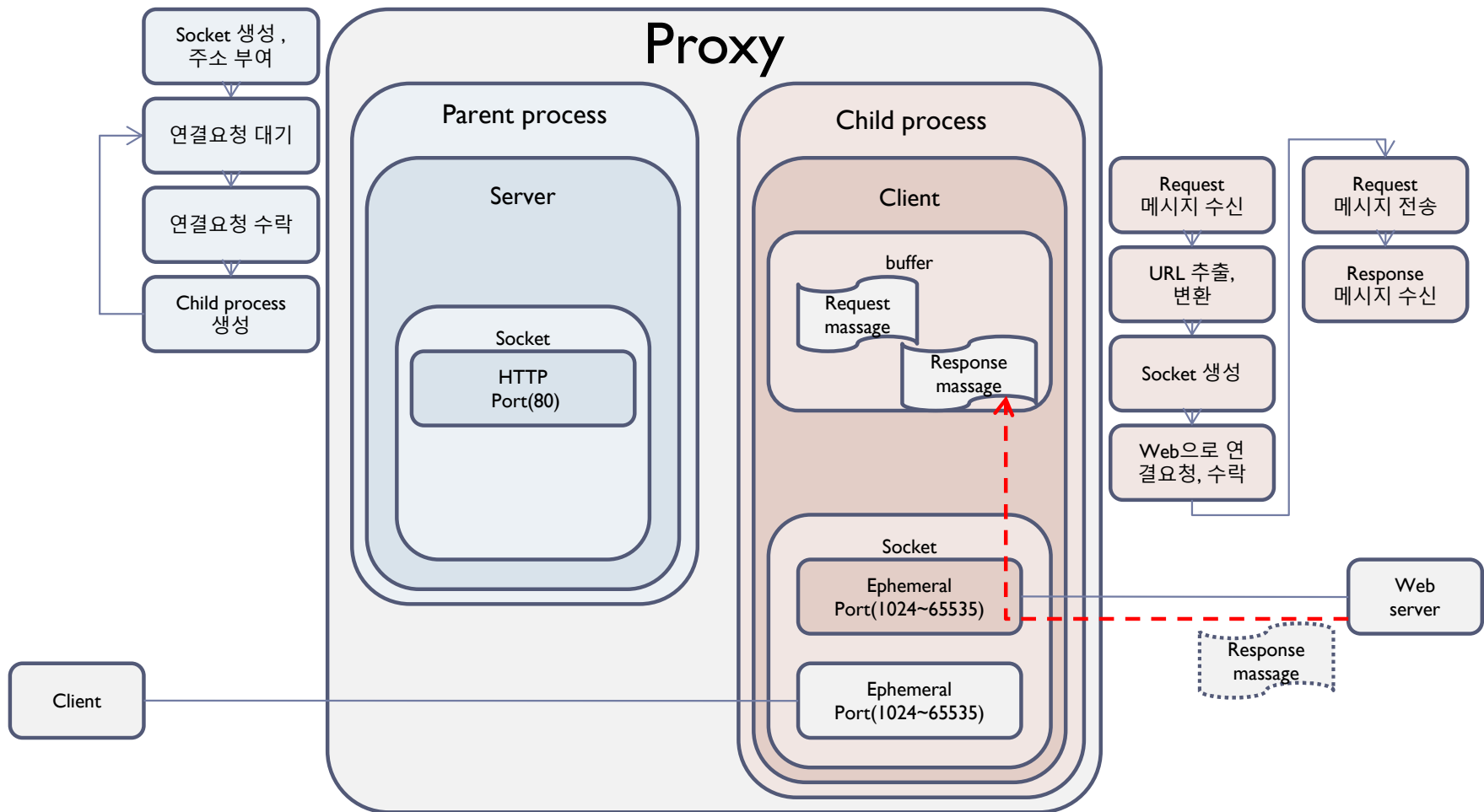
Proxy Connection (4/9)



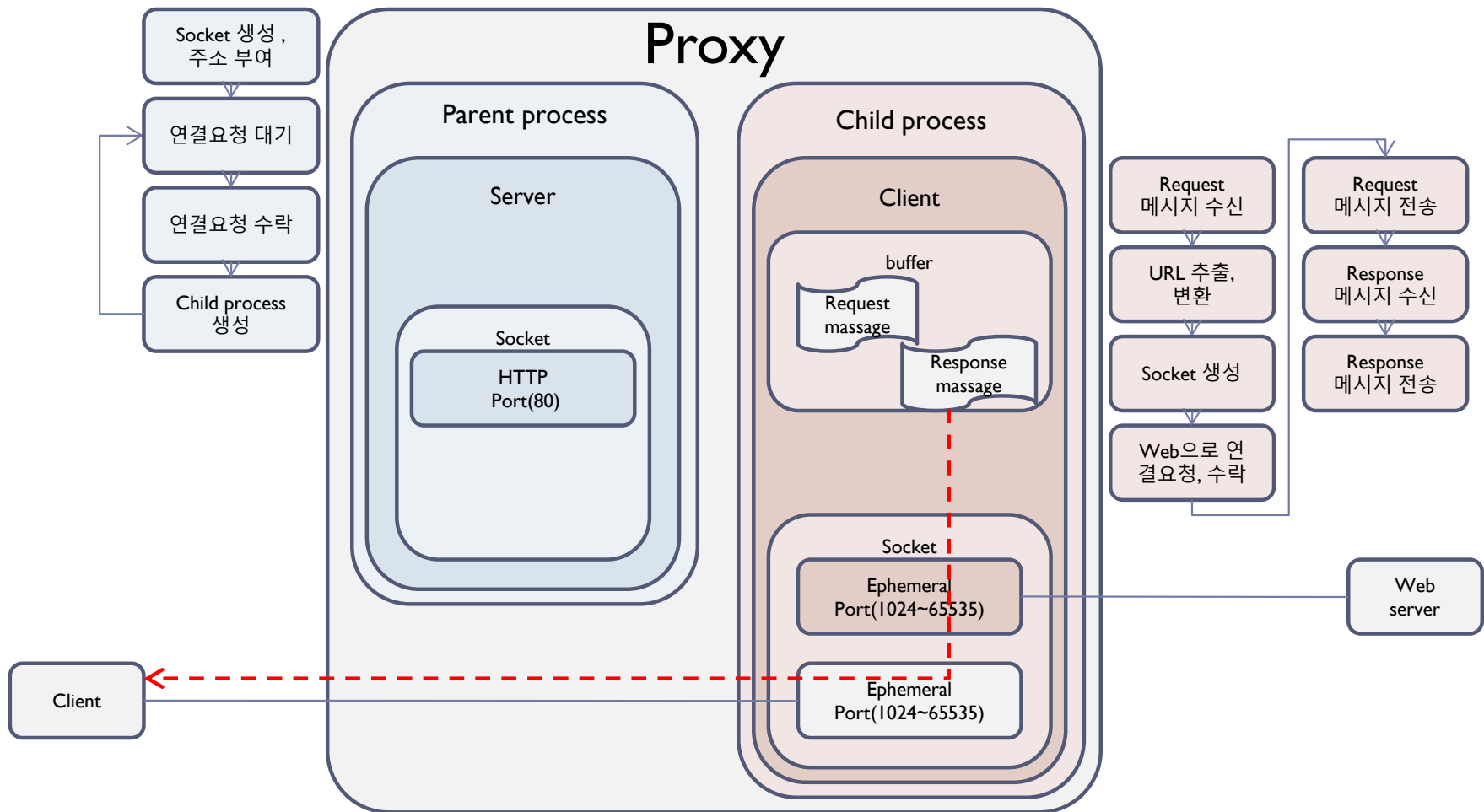
Proxy Connection (5/9)



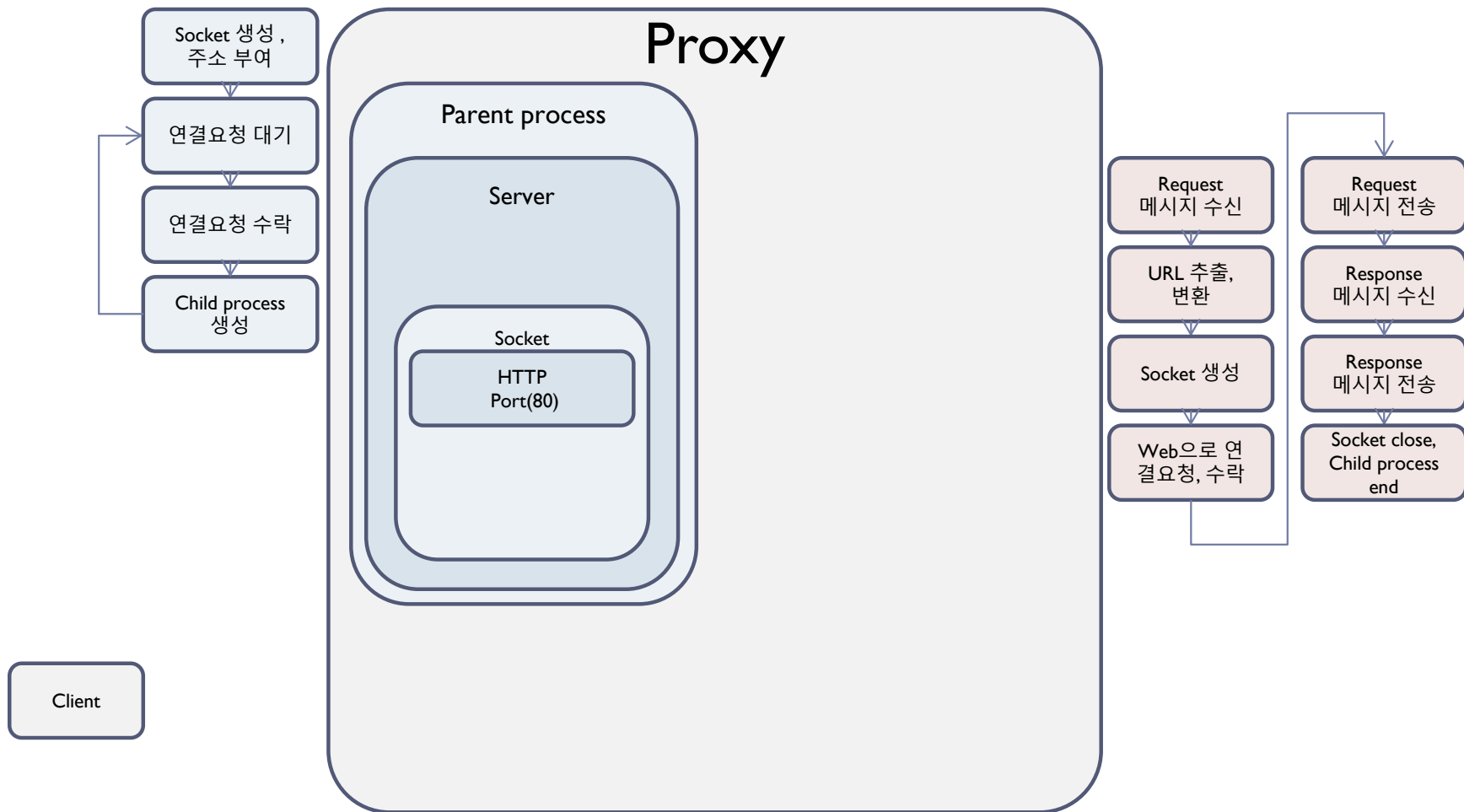
Proxy Connection (6/9)



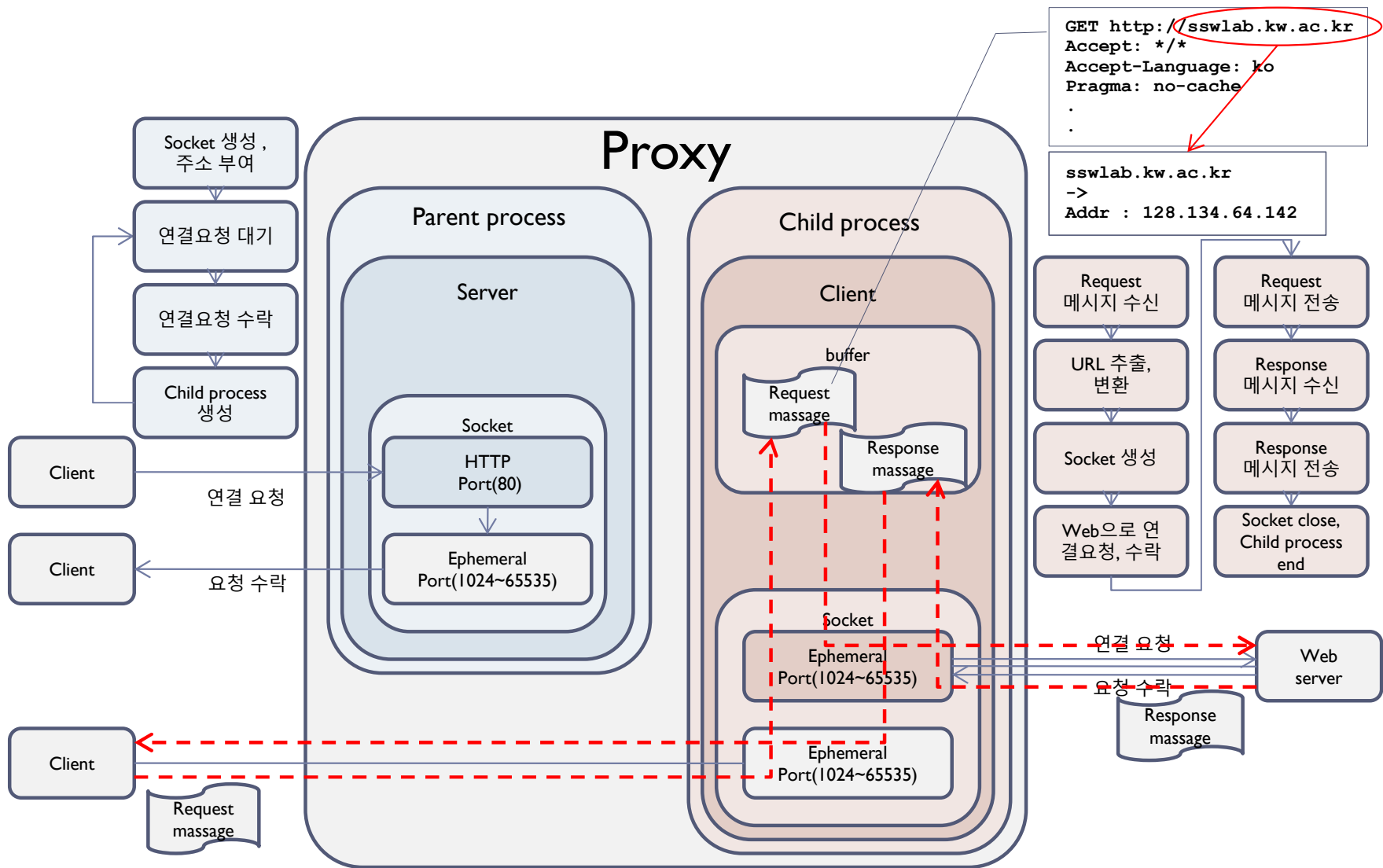
Proxy Connection (7/9)



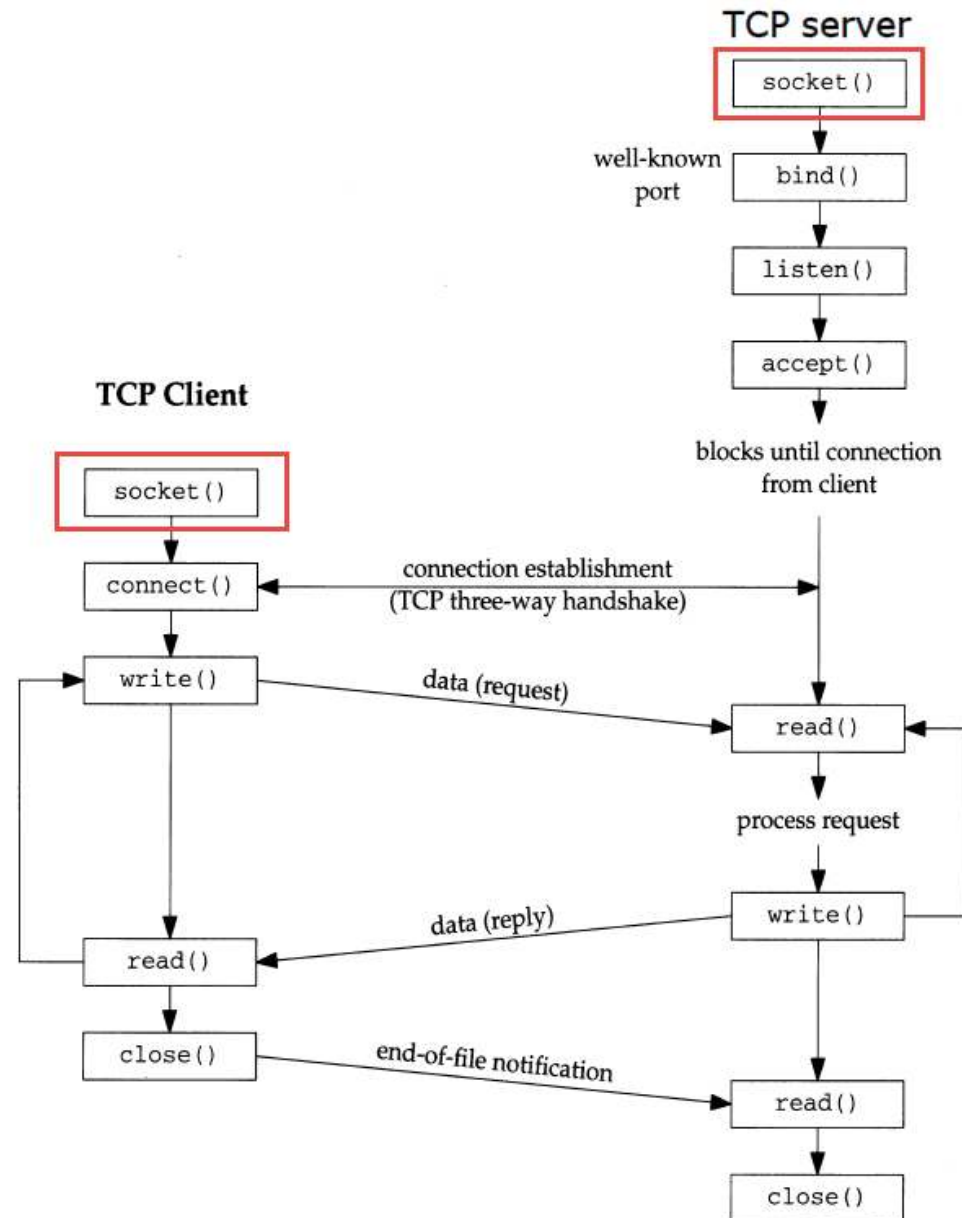
Proxy Connection (8/9)



Proxy Connection (9/9)

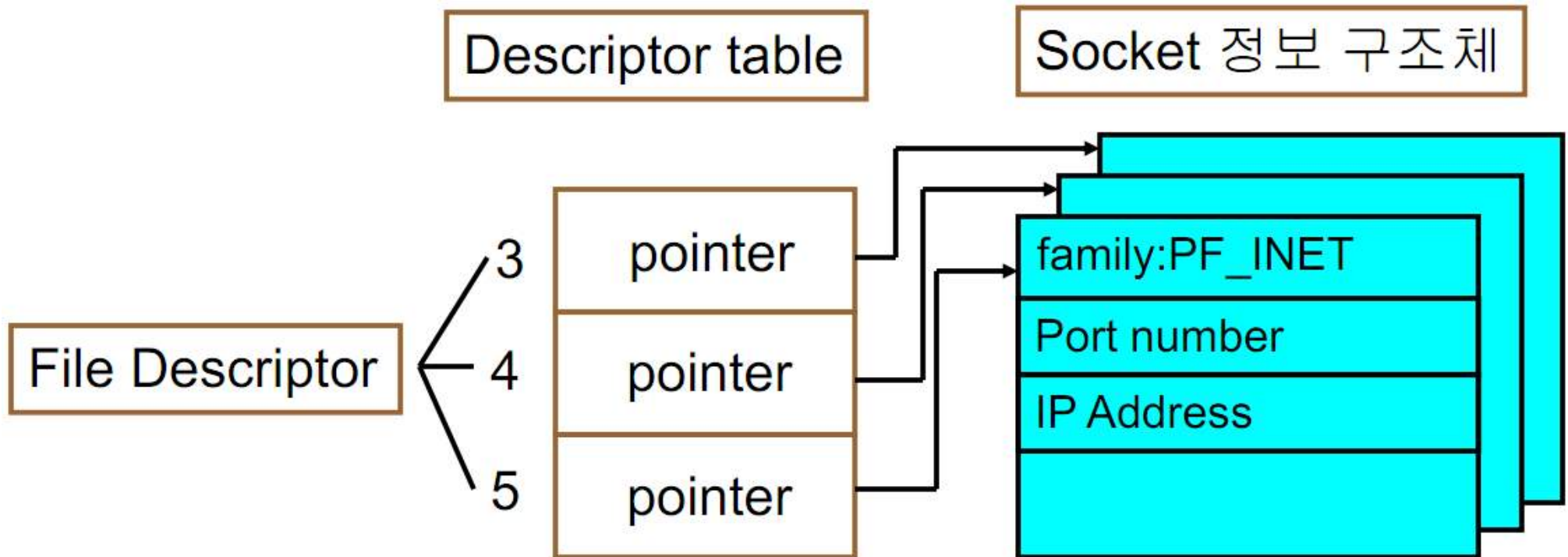


Socket Programming



Socket

- TCP/IP를 이용하는 API
- Socket number
 - Socket 개설 시 return되는 값



struct sockaddr_in

- **소켓 주소의 구조체**

- connection 설정 시 필요한 socket 정보로 구성

```
struct sockaddr_in
{
    short sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
}
```

- **sin_family**

- 주소 체계
- Ex) AF_INET : IPv4 인터넷 프로토콜 체계
 - PF_INET과 동일하나 address family에서는 AF_INET을 권장

- **sin_port**

- 16비트 포트 번호

- ▶ **sin_addr**

- ▶ 32비트 IP 주소

- ▶ **sin_zero[8]**

- ▶ struct sockaddr 과의 호환성을 위한 dummy
- ▶ 실제 사용되지는 않으며 sockaddr과 사이즈를 맞추기 위함

socket()

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

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

- Connection을 위한 시스템의 socket을 설정
- Returns: Nonnegative descriptor if OK, -1 on error
- domain: protocol 형태 (Protocol Family)
 - Ex) PF_INET : IPv4 인터넷 프로토콜 체계
 - AF_INET과 동일하나 protocol family에서는 PF_INET을 권장
- type: service type
 - Ex) SOCK_STREAM : TCP 방식
 - Ex) SOCK_DGRAM : UDP 방식
- protocol: 프로토콜
 - Ex) 0 : type 설정 값에 따라 자동으로 프로토콜 선택

bind()

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

```
int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);
```

- 소켓과 sockaddr_in 구조체 정보를 binding
 - 즉, 생성한 소켓을 사용할 수 있도록 ip, 포트 번호를 할당
 - Returns: 0 if OK, -1 on error
-
- sockfd : socket() 호출 시 리턴값 (소켓번호)
 - sockaddr : sockaddr struct (서버 자신의 소켓 주소 구조체 포인터)
 - addrlen : sockaddr struct의 size (*myaddr 구조체의 크기)

listen()

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

int listen(int sockfd, int backlog);
```

- Server에서만 사용하며, client로 부터 connection을 위해 대기상태(listen)로 변경
 - 연결요청이 들어오면 대기 queue에 넣어줌
 - Returns: 0 if OK, -1 on error
-
- sockfd : socket() 호출 시 리턴값 (소켓 번호)
 - backlog : 대기열 queue의 길이 (연결을 기다리는 클라이언트의 최대수)

accept()

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

```
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```

- 서버가 client의 접속을 받아들임
 - listen() 함수에 의해 대기 queue에 있는 연결요청들을 순서대로 수락
 - Returns: nonnegative descriptor if OK, -1 on error
-
- sockfd : socket()호출시 리턴값 (소켓번호)
 - sockaddr : sockaddr struct (연결요청을 한 상대방의 소켓주소 구조체)
 - addrlen : sockaddr struct의 size (addr 구조체 크기의 포인터)

connect()

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


int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);
```

- Client가 server와의 connection을 설정하기 위해 사용
 - Returns: nonnegative descriptor if OK, -1 on error
-
- sockfd : socket() 호출 시 리턴값 (서버와 연결시킬 소켓번호)
 - sockaddr : sockaddr struct (상대방 서버의 소켓주소 구조체)
 - addrlen : sockaddr struct의 size (구조체 *servaddr의 크기)

close()

```
#include <unistd.h>

int close(int filedes)
```

- File descriptor  close
- Returns: 0 if OK, -1 on error
- `filedes` : File descriptor

write()

```
#include <unistd.h>

ssize_t write(int fildes, const void * buf, size_t nbytes);
```

- 파일에 데이터를 출력(전송)하는 함수 (system call)
- Returns: 성공 시 전달 한 바이트 수, -1 on error
- fildes: 데이터 전송 영역을 나타내는 파일 디스크립터
- buf : 전송할 데이터를 가지고 있는 버퍼의 포인터
- nbytes : 전송할 데이터의 바이트 수

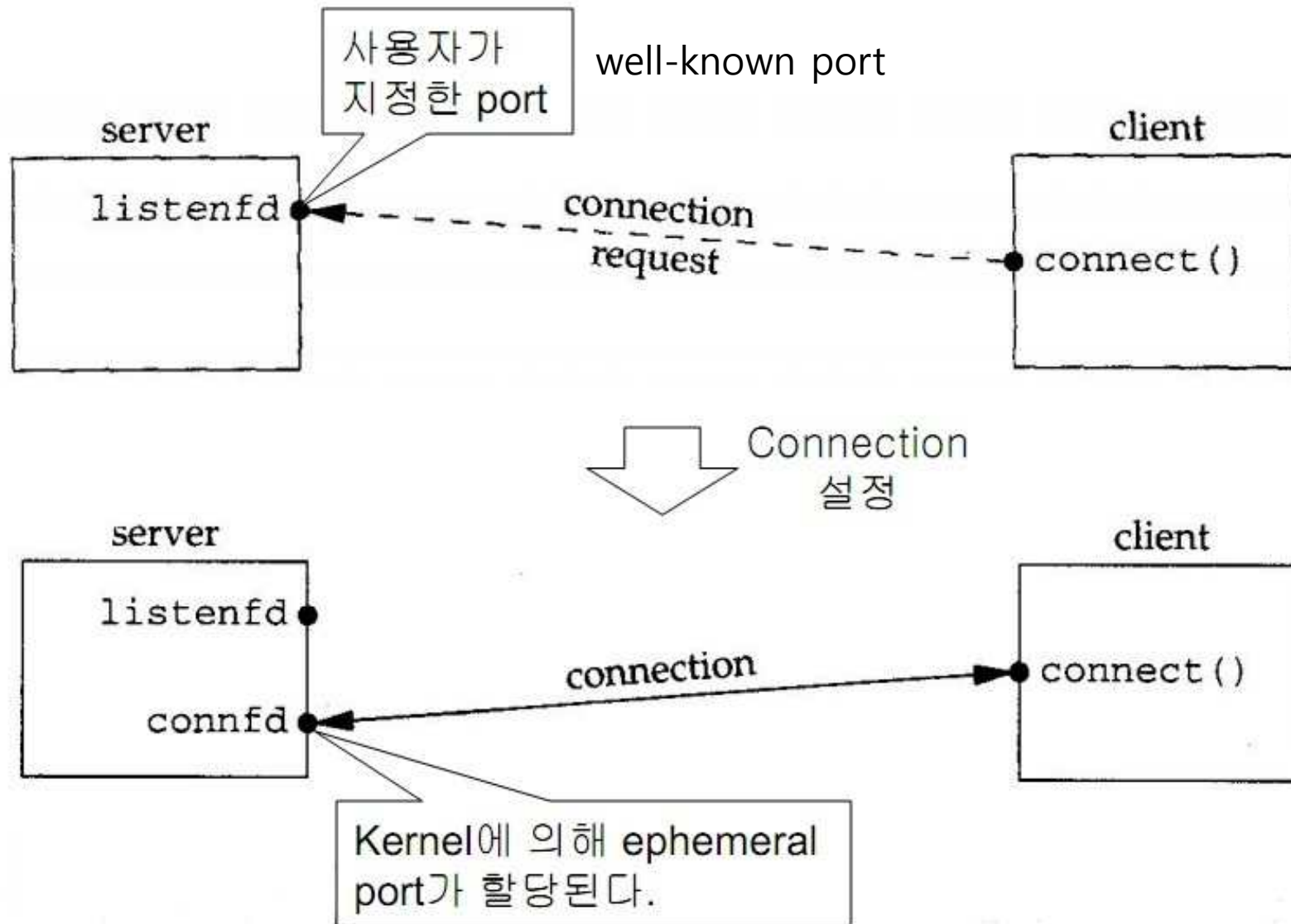
read()

```
#include <unistd.h>
```

```
ssize_t read(int fildes, void *buf, size_t nbytes);
```

- read 함수는 데이터를 입력(수신)받는 함수 (system call)
- Returns: 성공 시 수신 한 바이트 수(단 EOF 만나면 0), -1 on error
- fildes: 데이터를 전송해 주는 대상을 가리키는 파일 디스크립터
- buf : 수신 한 데이터를 저장할 버퍼를 가리키는 포인터
- nbytes : 수신할 최대 바이트 수
- ※ close(), write(), read() 함수는 소켓 통신에만 한정되지 않음

Status of Server and Client



htonl(), htons(), ntohl(), ntohs()

```
#include <netinet/in.h>

unsigned long int htonl(unsigned long int  hostlong);
unsigned short int htons(unsigned short int  hostshort);
unsigned long int ntohl(unsigned long int  netlong);
unsigned short int ntohs(unsigned short int  netshort);
```

- CPU마다 호스트 바이트 순서가 다르기 때문에 네트워크를 통한 데이터 전송 시 네트워크 바이트 순서(big endian)로 통일
- 호스트와 네트워크 간 바이트 순서를 바꿈
 - htonl : long data(IP addr.) to network byte order
 - htons : short data(port no.) to network byte order
 - ntohl : long data(IP addr.) to host byte order
 - ntohs : short data(port no.) to host byte order
- i80x86에서
 - 호스트 바이트 순서 – 하위 바이트가 앞 (little-endian) Ex) 192.168.0.1 저장 → 1.0.168.192
 - 네트워크 바이트 순서 – 상위 바이트가 앞 (big-endian) Ex) 192.168.0.1 저장 → 192.168.0.1

inet_addr()

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long int inet_addr(const char *cp);
```

- Dotted decimal 형태인 인터넷 호스트 주소를 네트워크 바이트 순서인 이진 데이터(32bit big-endian)로 바꿈
 - EX) 192.168.1.102 → 6601a8c0
- Returns
 - If the input is invalid, INADDR_NONE (usually -1) is returned
- Parameters
 - cp : Dotted decimal notation

Socket Programming APIs

기능	관련 API
자식 프로세스 생성	fork()
서버 역할 소켓 통신	socket(), setsockopt(), htonl(), htons(), bind(), listen(), accept(), close()
요청 문자열에서 HOST 부분 추출	strtok(), strcpy()
클라이언트 역할 소켓 통신	gethostbyname(), socket(), htons(), connect(), read(), write(), close()
웹 서버 “응답 없음” 처리	signal(), alarm(), raise()

Practice1. Echo Server – “server.c” (1/2)

```
1 #include <stdio.h>
2 #include <string.h>      // bzero(), ...
3 #include <sys/types.h>
4 #include <sys/socket.h>
5 #include <netinet/in.h>
6 #include <unistd.h>      // STDOUT_FILENO, ...
7
8 #define BUFFSIZE        1024
9 #define PORTNO           40000
10
11 int main()
12 {
13     struct sockaddr_in server_addr, client_addr;
14     int socket_fd, client_fd;
15     int len, len_out;
16     char buf[BUFFSIZE];
17
18     if ((socket_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
19         printf("Server: Can't open stream socket.");
20         return 0;
21     }
22
23     bzero((char *)&server_addr, sizeof(server_addr));
24     server_addr.sin_family = AF_INET;
25     server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
26     server_addr.sin_port = htons(PORTNO);
27     if (bind(socket_fd, (struct sockaddr *)&server_addr,
28             sizeof(server_addr)) < 0) {
29         printf("Server: Can't bind local address.\n");
30         return 0;
31     }
```


Practice1. Echo Server – “server.c” (2/2)

```
32
33     listen(socket_fd, 5);
34     while(1) {
35         len = sizeof(client_addr);
36         client_fd = accept(socket_fd,
37                             (struct sockaddr*)&client_addr, &len);
38         if(client_fd < 0) {
39             printf("Server: accept failed.\n");
40             return 0;
41         }
42
43         printf("[%d:%d] client was connected.\n",
44               client_addr.sin_addr.s_addr, client_addr.sin_port);
45         while( ( len_out = read(client_fd, buf, BUFSIZE) ) > 0 ) {
46             write(STDOUT_FILENO, "  - Messages : ", 15);
47             write(STDOUT_FILENO, buf, len_out);
48             write(client_fd, buf, len_out);
49             write(STDOUT_FILENO, "\n", 1);
50         }
51         printf("[%d:%d] client was disconnected.\n",
52               client_addr.sin_addr.s_addr, client_addr.sin_port);
53         close(client_fd);
54     }
55     close(socket_fd);
56     return 0;
57 }
58
```

Practice1. Echo Client – “client.c” (1/2)

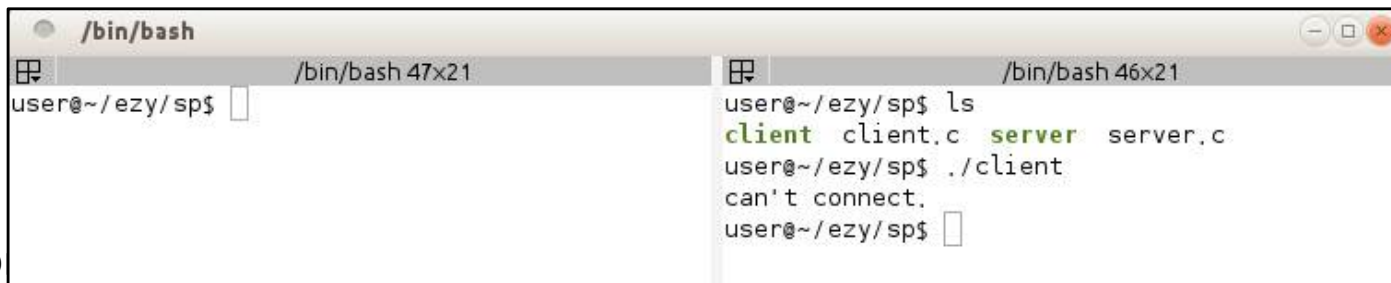
```
1 #include <stdio.h>
2 #include <string.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/socket.h>
6 #include <netinet/in.h>
7
8 #define BUFFSIZE      1024
9 #define PORTNO        40000
10
11 int main()
12 {
13     int socket_fd, len;
14     struct sockaddr_in server_addr;
15     char haddr[] = "127.0.0.1";
16     char buf[BUFFSIZE];
17
18     if( (socket_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0 ) {
19         printf("can't create socket.\n");
20         return -1;
21     }
22
23     bzero( buf, sizeof(buf) );
24     bzero( (char*)&server_addr, sizeof(server_addr) );
25     server_addr.sin_family = AF_INET;
26     server_addr.sin_addr.s_addr = inet_addr(haddr);
27     server_addr.sin_port = htons(PORTNO);
28
```

Practice1. Echo Client – “client.c” (2/2)

```
29     if( connect(socket_fd, (struct sockaddr*)&server_addr,
30                  sizeof(server_addr)) < 0 ) {
31         printf("can't connect.\n");
32         return -1;
33     }
34
35     write(STDOUT_FILENO, "> ", 2);
36     while ( (len = read(STDIN_FILENO, buf, sizeof(buf))) > 0 ) {
37         if (write(socket_fd, buf, strlen(buf)) > 0) {
38             if ( (len = read(socket_fd, buf, sizeof(buf))) > 0 ) {
39                 write(STDOUT_FILENO, buf, len);
40                 bzero( buf, sizeof(buf) );
41             }
42         }
43         write(STDOUT_FILENO, "> ", 2);
44     }
45     close(socket_fd);
46     return 0;
47 }
```

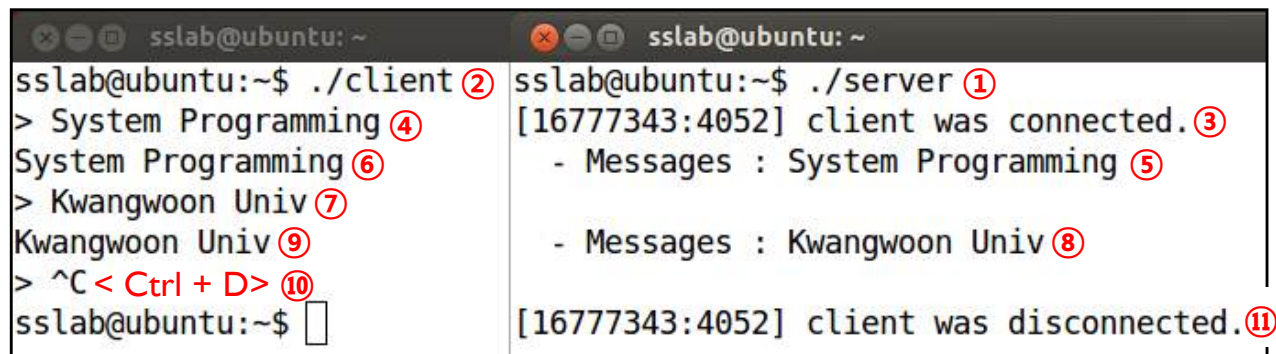
Practice1. Execution of Echo Server and Client

- Command 창은 두 개를 이용
- server를 실행하지 않은 상태에서 client 실행 시 **can't connect**.



The image shows two terminal windows. The left window, titled '/bin/bash 47x21', shows a user prompt 'user@~/ezy/sp\$' with a cursor. The right window, titled '/bin/bash 46x21', shows the same user prompt followed by the command 'ls' which lists 'client.c' and 'server.c'. Then, the user runs './client', and the output is 'can't connect.' followed by another user prompt.

- 하나의 co
- Client가 정상 실행되면 > 문자가 출력되고 메시지를 입력하면 server로 전달
 - Server는 받은 메시지를 client에게 다시 전달
 - Client는 결국 자신이 보낸 메시지를 받아 출력.



The image shows two terminal windows side-by-side. The left window, titled 'sslab@ubuntu: ~', shows the user running './client' (labeled 2), which prompts for input. The user enters 'System Programming' (labeled 4), which is echoed back (labeled 6). Then the user enters 'Kwangwoon Univ' (labeled 7), which is echoed back (labeled 9). Finally, the user presses Ctrl+C (labeled 10) to exit. The right window, also titled 'sslab@ubuntu: ~', shows the user running './server' (labeled 1). The server outputs '[16777343:4052] client was connected.' (labeled 3). It then receives the message 'System Programming' (labeled 5) and echoes it back (labeled 5). Next, it receives 'Kwangwoon Univ' (labeled 8) and echoes it back (labeled 8). Finally, it outputs '[16777343:4052] client was disconnected.' (labeled 11).

※ Server 종료 후 다시 실행할 때
'Server: Can't bind local address.' 라는
메시지가 뜨면, 1~2분 정도 후 다시 실행하면 됨

Practice2. Multi Echo Server – “server.c” (1/3)

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>

#define BUFFSIZE      1024
#define PORTNO        40000

static void handler()
{
    pid_t pid;
    int status;
    while((pid = waitpid(-1, &status, WNOHANG)) > 0);
}

int main()
{
    struct sockaddr_in server_addr, client_addr;
    int socket_fd, client_fd;
    int len, len_out;
    int state;
    char buf[BUFFSIZE];
    pid_t pid;

    if ((socket_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Server : Can't open stream socket\n");
        return 0;
    }

    bzero((char*)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    server_addr.sin_port = htons(PORTNO);
```


Practice2. Multi Echo Server – “server.c” (2/3)

```
if (bind(socket_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0)
{
    printf("Server : Can't bind local address\n");
    close(socket_fd);
    return 0;
}

listen(socket_fd, 5);
signal(SIGCHLD, (void *)handler);

while (1)
{
    bzero((char*)&client_addr, sizeof(client_addr));
    len = sizeof(client_addr);
    client_fd = accept(socket_fd, (struct sockaddr*)&client_addr, &len);

    if (client_fd < 0)
    {
        printf("Server : accept failed  %d\n", getpid());
        close(socket_fd);
        return 0;
    }

    printf("[%d : %d] client was connected\n", client_addr.sin_addr.s_addr, client_addr.sin_port);
    pid=fork();

    if(pid==-1)
    {
        close(client_fd);
        close(socket_fd);
        continue;
    }
}
```

Practice2. Multi Echo Server – “server.c” (3/3)

```
if(pid==0)
{
    while ((len_out = read(client_fd, buf, BUFFSIZE)) > 0)
    {
        if(!strncmp(buf, "bye", 3))
        {
            break;
        }
        write(STDOUT_FILENO, " - Messages : ", 15);
        write(STDOUT_FILENO, buf, len_out);
        write(client_fd, buf, len_out);
        write(STDOUT_FILENO, "\n", 1);
    }
    printf("[%d : %d] client was disconnected\n", client_addr.sin_addr.s_addr, client_addr.sin_port);
    close(client_fd);
    exit(0);
}
close(client_fd);
}
close(socket_fd);

return 0;
```

```
}
```

Practice2. Execution of Echo Server and Multiple Client

- Command 창은 여러 개를 이용
- 하나의 command 창에서는 server 실행, 다른 여러 개의 command 창에서는 다수 개의 client를 실행
- Client가 실행되면 server는 sub server process를 생성
 - Sub server는 client의 메시지를 받아 client에게 다시 전달
- Client가 정상 실행되면 > 문자가 출력되고 메시지를 입력하면 server로 전달
 - ➔ Server는 받은 메시지를 client에게 다시 전달
 - ➔ Client는 결국 자신이 보낸 메시지를 받아 출력.

```
sslab@ubuntu:~$ ./Server
[16777343 : 51361] client was connected
- Messages : Hello
[16777343 : 51873] client was connected
- Messages : Kwangwoon University
[16777343 : 51361] client was disconnected
[16777343 : 51873] client was disconnected
^C
sslab@ubuntu:~$
```

```
sslab@ubuntu:~$ ./Client
> Hello
Hello
> bye
sslab@ubuntu:~$
```

```
sslab@ubuntu:~$ ./Client
> Kwangwoon University
Kwangwoon University
> bye
sslab@ubuntu:~$
```


2025년 1학기 시스템프로그래밍 & 시스템프로그래밍실습

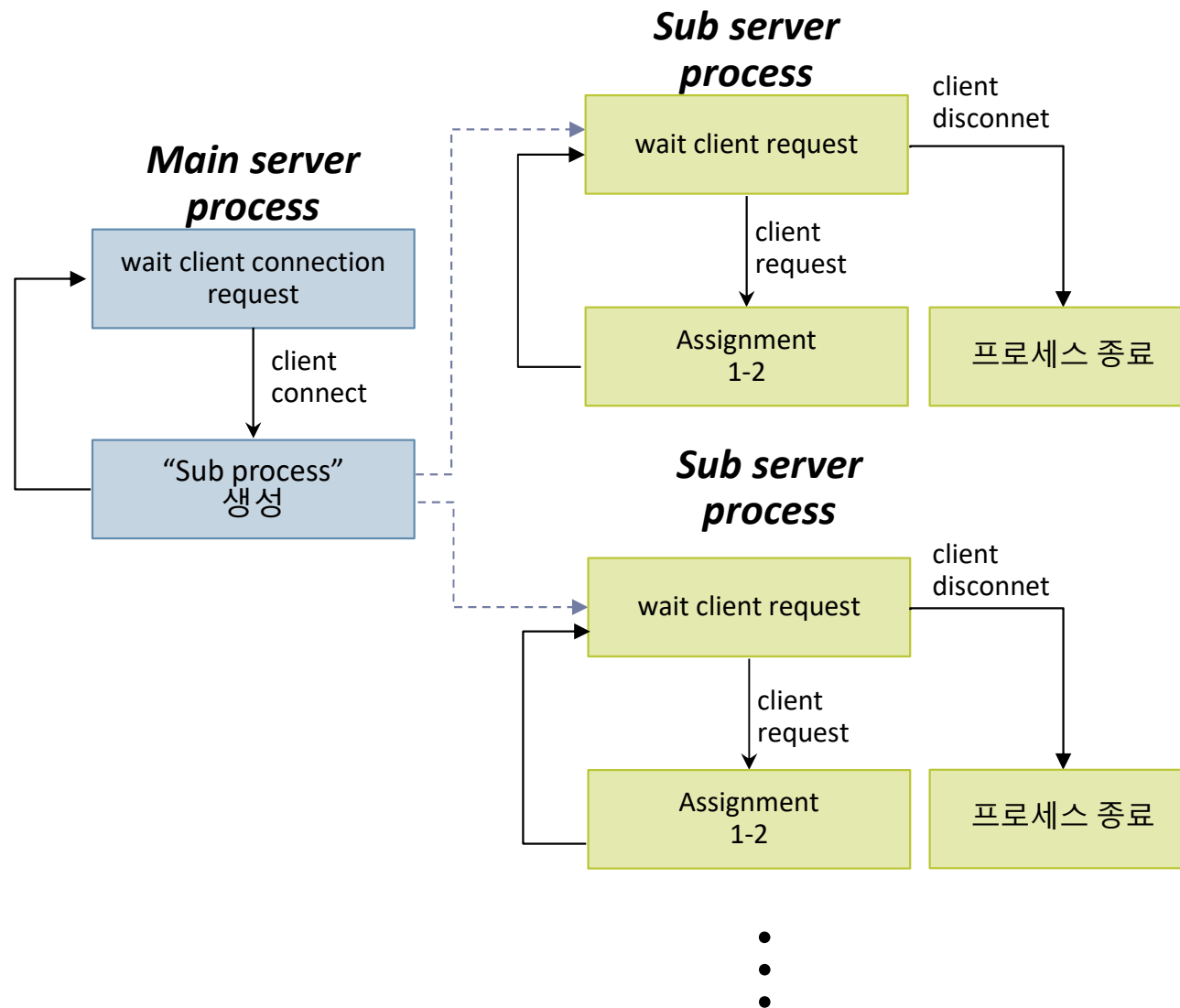
Proxy #2-1

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

2nd Assignment's Descriptions

- **Assignment 2-1**
 - Implement server/client
- **Assignment 2-2**
 - HTTP request handling in proxy server
- **Assignment 2-3**
 - Forward HTTP request to web server and signal handling
- **Assignment 2-4**
 - Add cache and log to proxy server

proxy 2-1(1/4)



proxy 2-1(2/4)

- 다수의 client를 처리하는 server와 client를 구현
- Server process
 - Main server process
 - client로부터의 통신 요청을 수락 후 client의 URL request 처리를 위한 새로운 프로세스("Sub process")를 생성하고 관리
 - 동작
 - client로부터 통신 요청이 들어오면 Sub process 생성
 - 이후 client로부터의 통신 요청을 대기
 - Sub server process
 - Client로부터 URL request를 받아 Assignment 1-2의 연산을 수행
 - 동작
 - Client connect와 disconnect를 알리는 정보를 아래와 같이 출력
 - [client IP : client port number] client was connected
 - [client IP : client port number] client was disconnected
 - client IP는 127.0.0.1로 표기
 - Main server process에서 출력해도 무방
 - Client로부터 URL을 전송 받음
 - Proxy 1-2의 연산을 수행
 - Hashing, Check **HIT or MISS(결과를 client에게 전송)**, Manipulate cache directory, Logging
 - 이후 client로부터의 URL 요청을 대기, client disconnect 시 프로세스 종료

proxy 2-1(3/4)

- 다수의 client를 처리하는 server와 client를 구현
- **Client process**
 - 사용자로부터 URL을 입력 받아 연결 되어있는 Sub server process로 URL 전송
 - 동작
 - "input URL> "를 출력하고, 사용자의 URL 입력을 대기
 - URL을 sub server process로 전송하고 HIT or MISS 결과를 sub server process로부터 받아서 출력
 - "bye" 입력 시 disconnect
 - client 종료
- **Log file**
 - Hit 일 경우
 - [HIT] ServerPID : pid | Directory name/file name-[Time]
 - [HIT] URL
 - Miss일 경우
 - [MISS] ServerPID : pid | URL-[Time]
 - [Time] : year/month/day, hour:min:sec 으로 표기
 - pid는 getpid()
 - "bye" 입력 시
 - 프로그램 실행 시간 기록, Hit, Miss 횟수 기록
 - e.g.
 - [Terminated] ServerPID : pid | run time : 7sec. #request hit : 2, miss : 3
 - pid : getpid()

proxy 2-1(4/4)

- Operation Example

sslslab@ubuntu:~\$./Server [127.0.0.1 : 16094] client was connected [127.0.0.1 : 16606] client was connected [127.0.0.1 : 16606] client was disconnected [127.0.0.1 : 16094] client was disconnected ^C sslslab@ubuntu:~\$ < Server >	sslslab@ubuntu:~\$./Client input url > www.kw.ac.kr MISS input url > www.google.com MISS input url > www.naver.com HIT input url > bye sslslab@ubuntu:~\$ < Client [127.0.0.1:16094] >	sslslab@ubuntu:~\$./Client input url > www.naver.com MISS input url > www.kw.ac.kr HIT input url > bye sslslab@ubuntu:~\$ < Client [127.0.0.1:16606] >
--	---	---

- Log file

sslslab@ubuntu:~\$ cat ./logfile/logfile.txt [MISS] ServerPID : 13232 www.kw.ac.kr - [2025/4/3, 04:47:58] [MISS] ServerPID : 13234 www.naver.com - [2025/4/3, 04:48:08] [HIT] ServerPID : 13234 e00/0f293fe62e97369e4b716bb3e78fababf8f90 - [2025/4/3, 04:48:10] [HIT]www.kw.ac.kr [Terminated] ServerPID : 13234 run time : 7 sec. #request hit : 1, miss : 1 [MISS] ServerPID : 13232 www.google.com - [2025/4/3, 04:48:46] [HIT] ServerPID : 13232 fed/818da7395e30442b1dcf45c9b6669d1c0ff6b - [2025/4/3, 04:48:57] [HIT]www.naver.com [Terminated] ServerPID : 13232 run time : 72 sec. #request hit : 1, miss : 2

Report Requirements

- **Ubuntu 20.04.6 Desktop 64bits 환경에서 채점**
- **Copy 발견 시 0점 처리**
- **보고서 구성**
 - **보고서 표지**
 - 수업 명, 과제 이름, 담당 교수님, 학번, 이름, 강의 시간 필히 명시
 - 과제 이름 → Proxy 2-1
 - **아래의 내용은 보고서에 필히 포함**
 - Introduction
 - 과제 소개 – 4줄 이상(background 제외) 작성
 - Flow Chart
 - 코드 작성 순서도
 - Pseudo code
 - 알고리즘
 - 결과화면
 - 수행한 내용을 캡처 및 설명
 - 고찰
 - 과제를 수행하면서 느낀 점 작성
 - Reference
 - 과제를 수행하면서 참고한 내용을 구체적으로 기록
 - 강의자료만 이용한 경우 생략 가능

Report Requirements

- **Softcopy Upload**

- **제출 파일**

- 보고서 + 소스파일 하나의 압축 파일로 압축하여 제출(tar.xz)
 - 1)보고서:
 - 보고서를 **pdf로 변환**하여 제출
 - 보고서 이름은 *Proxy2-1_수강분류코드_학번_이름* 으로 작성
 - 2)C 파일 명:
 - **server.c, client.c**
 - **Comment 작성(Appendix 내용 참고)**
 - 3)Makefile:
 - 실행파일명: **server, client**
 - C 파일명, 실행파일명 지정한 이름 외 다른 명으로 작성 시 감점


- **tar.xz 압축 방법**

- **(Appendix 내용 참고)**

- 컴파일은 무조건 Makefile(makefile)을 이용한 make로 함.
 - Makefile(makefile) 없거나 실행 불가시 0점
 - 파일 압축 오류 시, 0점 처리

Report Requirements

- 실습 수업을 수강하는 학생인 경우
 - 실습 과목에 과제를 제출(.tar.xz)
 - 이론 과목에 간단한 .txt 파일로 제출

 실습수업때 제출했습니다.

2022-08-29 오후 3:58

텍스트 문서

0KB

- 이론 과목에 .txt 파일 미 제출 시 감점
- .tar.xz 파일로 제출 하지 않을 시 감점

- 예시-이론 월5 수6 수강하는 학생인 경우
 - 보고서: Proxy2-1_A_2025123456_홍길동.pdf
 - 압축 파일 명: Proxy2-1_A_2025123456_홍길동.tar.xz

수강요일	이론1 월5수6	이론2 목4	실습1 목12
수강분류 코드	A	B	C

- 과제 제출
 - KLAS – 강의 과제 제출
 - 2025년 5월 1일 목요일 23:59까지 제출
 - 딜레이 받지 않음
 - 제출 마감 시간 내 미제출시 해당 과제 0점 처리
 - 교내 서버 문제 발생 시, 메일로 과제 제출 허용