TCP 소켓 프로그래밍

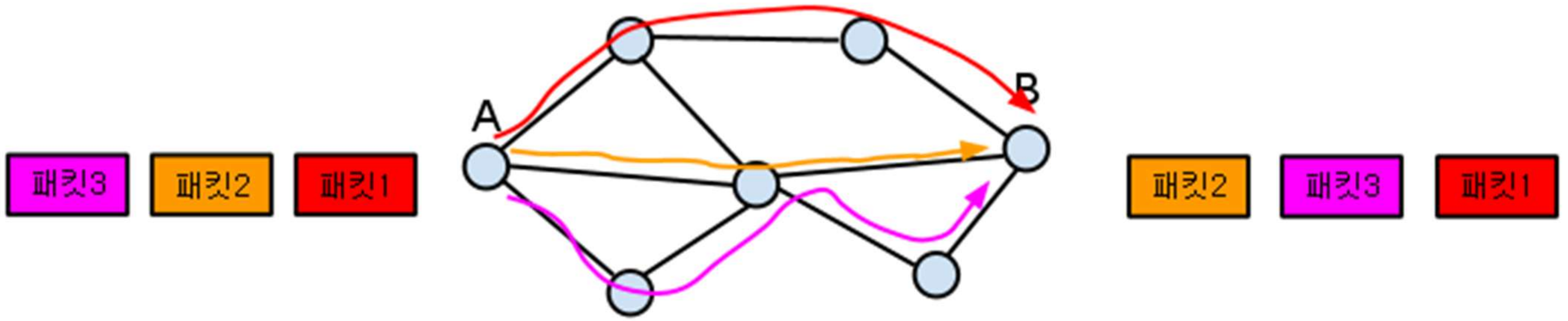
뇌를 자극하는 TCP/IP 소켓 프로그래밍

TCP 소켓 프로그래밍

- 인터넷 : 프로그램과 프로그램의 연결
- 연결 방식의 차이에 의한 프로토콜
  - TCP : 연결 지향(connection-oriented) 프로토콜
  - UDP : 데이터그램(datagram) 프로토콜

패킷 통신

- 인터넷에서 통신은 패킷 형태로 이루어진다.
- 목적지 까지 다양한 경로가 가능하다
  - 패킷 순서 변경
- 패킷 누락 가능



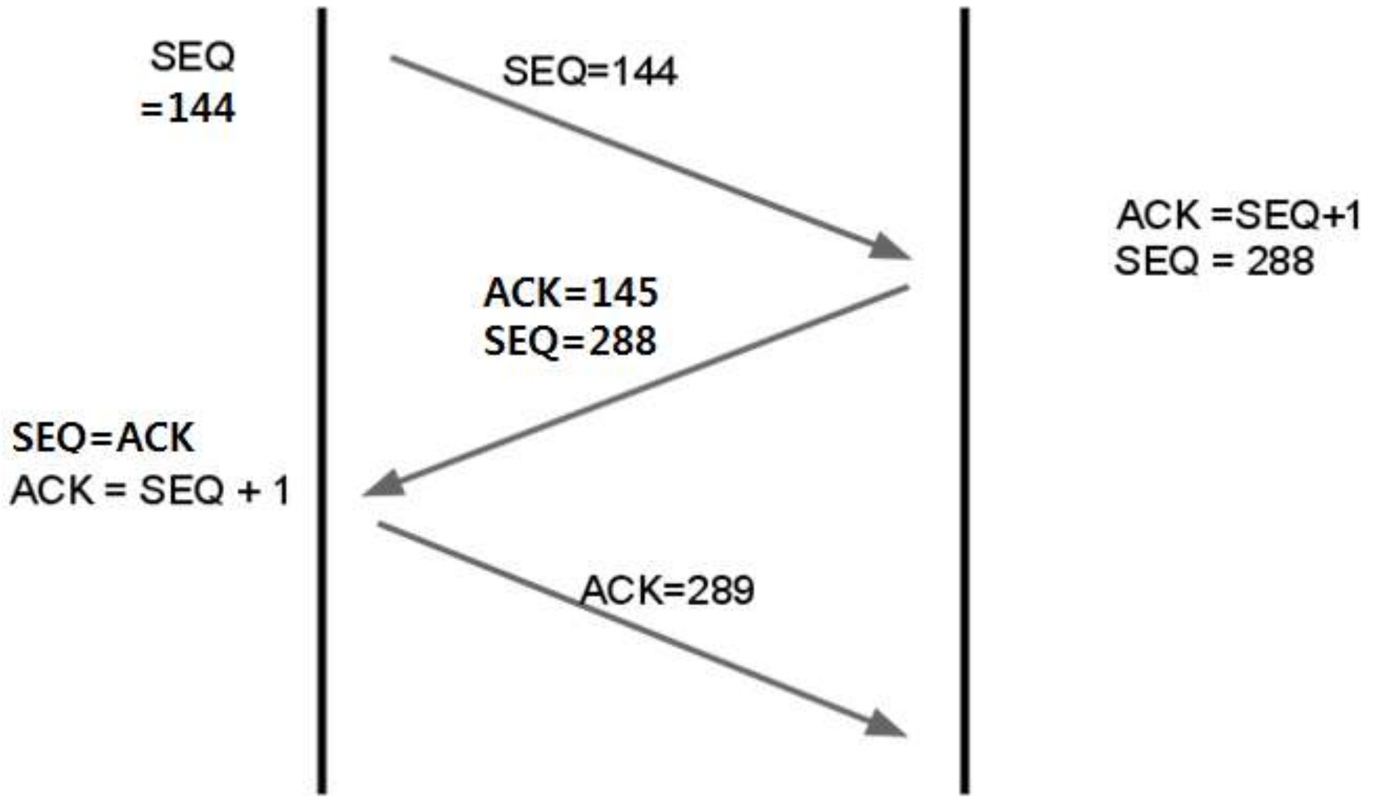
패킷 통신과 TCP

- 서버와 클라이언트 간에 전용 선로 개설
  - 연결 지향(connection-oriented)

- 신뢰성 있는 데이터 통신(reliable data communication)
  - 패킷의 순서 재 조립(reassembly)
  - 패킷 누수시 재 전송(retransmission)

TCP의 특징 : 연결지향

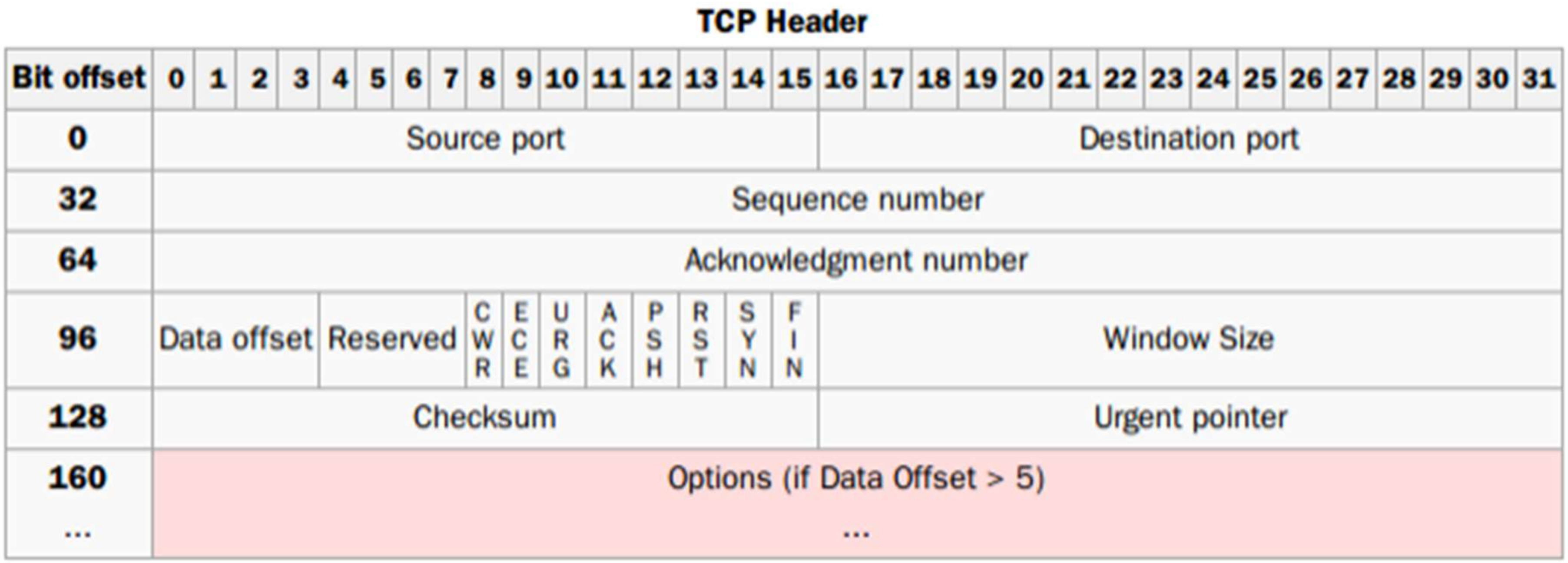
- 연결 지향 : 전용 선로 개설, 세션(session) 생성
- **Three-way handshake**
  - 3번의 패킷 교환으로 세션 개설



TCP의 특징 : 신뢰성 있는 데이터 교환

- 모든 패킷 전송에 대한 응답 확인
- 패킷에 일련번호 : 패킷을 흐름으로 다룸

TCP 헤더



TCP Header

Bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Source port																Destination port															
32	Sequence number																															
64	Acknowledgment number																															
96	Data offset				Reserved				CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
128	Checksum																Urgent pointer															
160 ...	Options (if Data Offset > 5) ...																															

- Source Port : 출발지 포트
- Destination Port : 목적지 포트
- Sequence number : 패킷 일련번호
- Acknowledgment number : ACK 번호
- Data offset : TCP헤더의 길이. User Data의 시작위치를 찾을 수 있다.
- Reserved
- Flags : 패킷의 Type을 알려주는 Flag
- Window Size : Receive window의 크기
- CheckSum : 패킷 error 체크
- Urgent Pointer : URG Flag가 On일 때, URG 데이터의 위치를 가리킨다.

TCP 소켓 프로그램 개발

- TCP 속성의 소켓 생성

```
socket(AF_INET, SOCK_STREAM, 0);
```

protocol

(Input) The protocol to be used on the socket. Supported values are:

0	Indicates that the default protocol for the type selected is to be used. For example, IPPROTO_TCP is chosen for the protocol if the type was set to SOCK_STREAM and the address family is AF_INET.
IPPROTO_IP	Equivalent to specifying the value zero (0).
IPPROTO_TCP	Indicates that the TCP protocol is to be used.
IPPROTO_UDP	Indicates that the UDP protocol is to be used.
IPPROTO_RAW	Indicates that communications is to the IP layer.
IPPROTO_ICMP	Indicates that the Internet Control Message Protocol (ICMP) is to be used.

TCP 소켓 프로그램 개발

- 서버측 소켓 설정

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
addr.sin_port = htons(8080)
bind(sockfd, (struct sockaddr *)&addr, sizeof(addr));
```

- INADDR_ANY : 0.0.0.0
  - BIND 할 주소

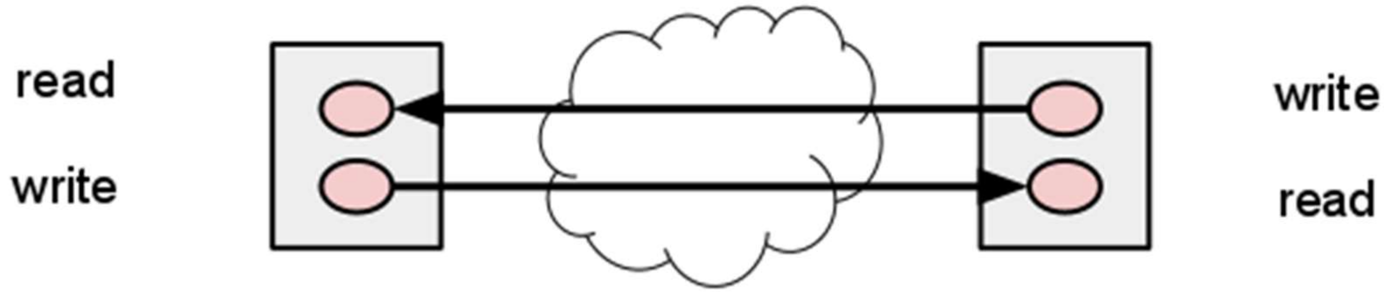
TCP 소켓 프로그램 개발

- 클라이언트측 설정

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("192.168.1.2");
addr.sin_port = htons(8080)
connect(sockfd, (struct sockaddr *)&addr, sizeof(addr));
```

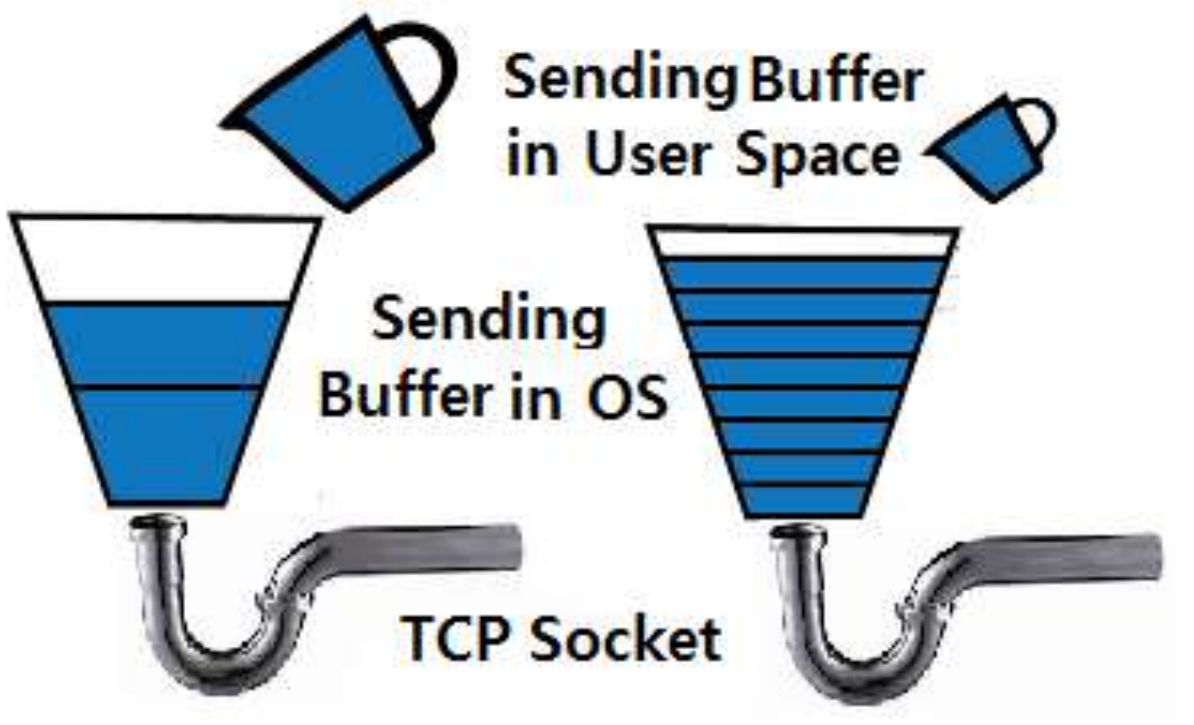
TCP 소켓 프로그램 개발

- 이후 데이터 통신은 **TCP**를 따른다.
  - 일반적인 입출력 함수를 이용 데이터 통신

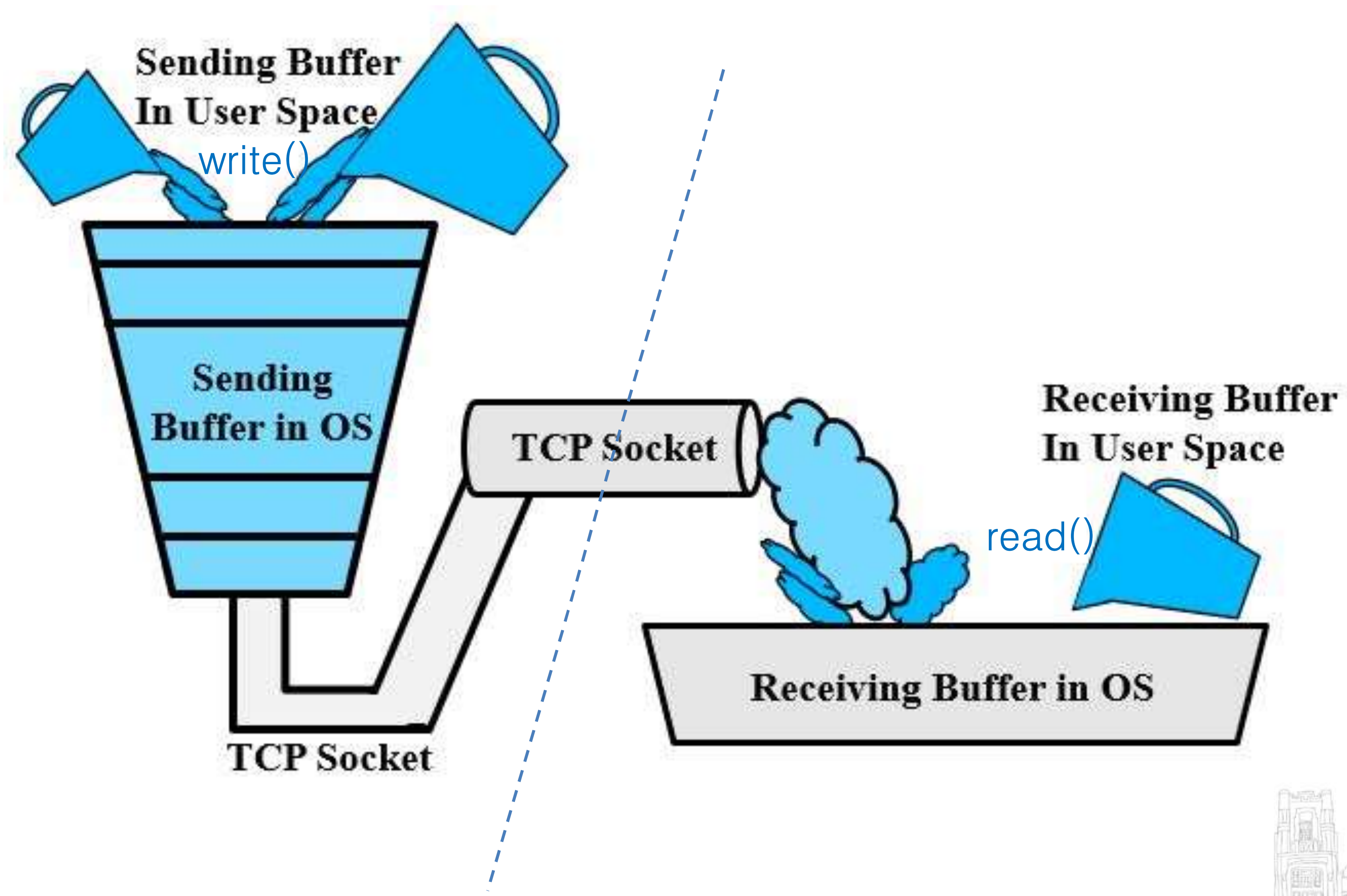


Buffers in TCP Stream(Connection-Oriented Data Transfer)

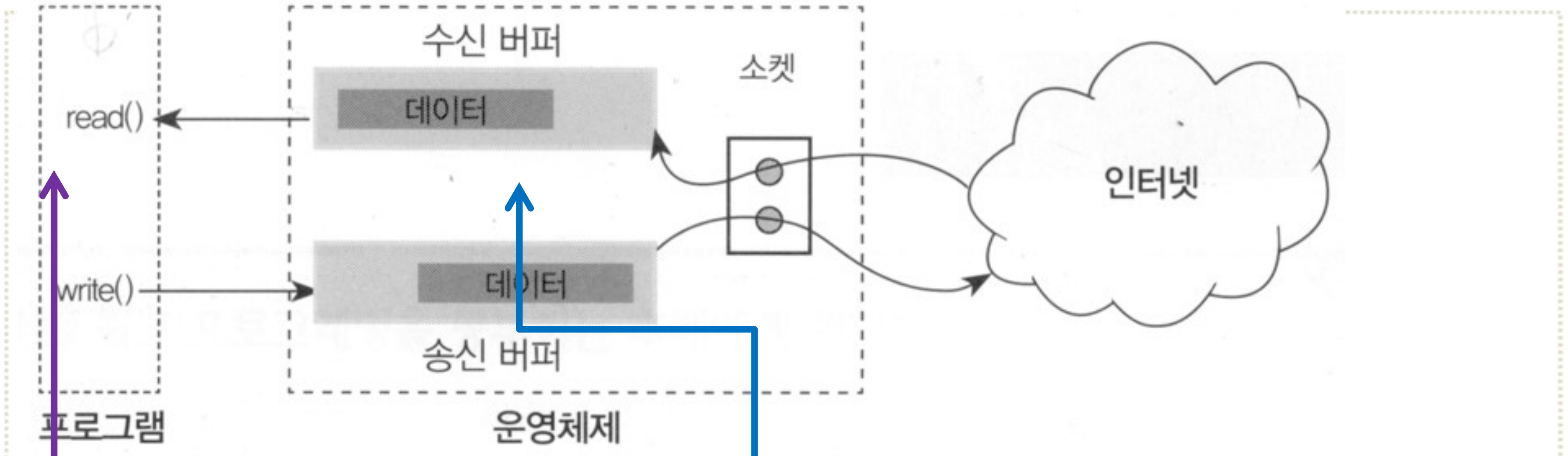
Buffers in TCP Stream(Connection-Oriented Data Transfer)



Buffers in TCP Stream(Connection-Oriented Data Transfer)



Buffers in TCP Stream(Connection-Oriented Data Transfer)



- 운영체제내 송신/수신 버퍼
- 응용 프로그램(user space)내의 user buffer(사용자 버퍼)

```
#define MAXLINE 256
char buf[MAXLINE];

memset(buf, 0x00,MAXLINE);
read(fd, buf, MAXLINE);
```

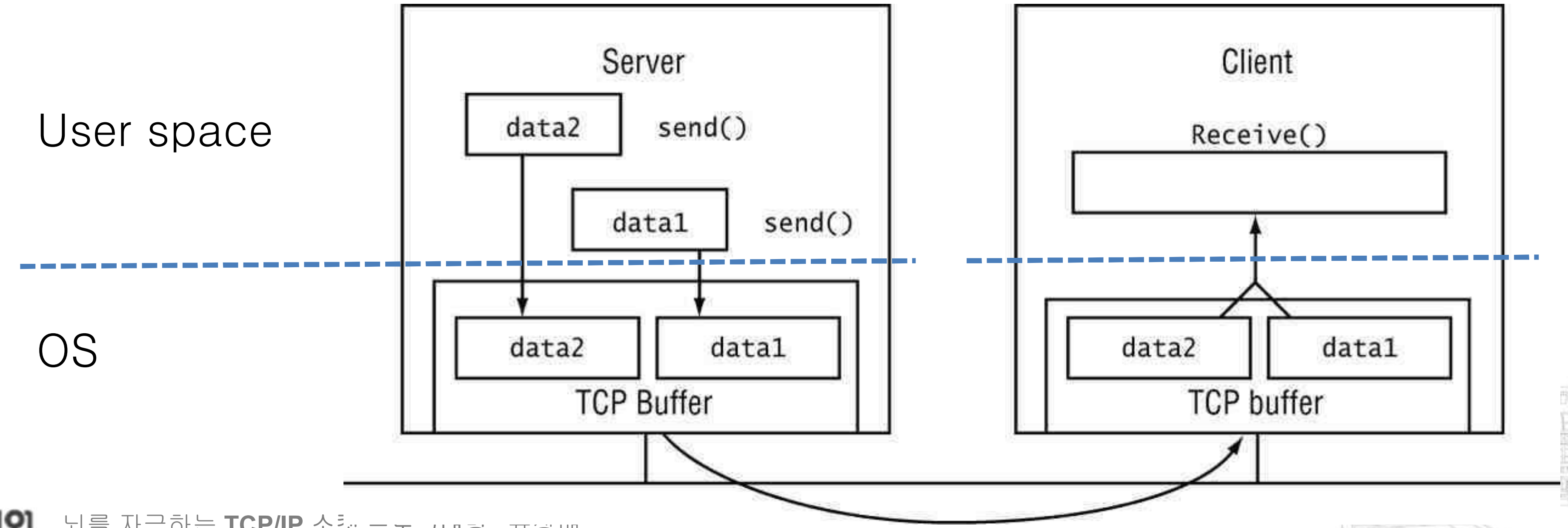
When TCP(Connection-oriented Data Transfer) Goes Bad

- Improper Data Buffer Handling in Programs
  - Receiver may not know the size of the sending buffer in user space(i.e., the size of the incoming data).

    •You must take care to ensure that all of the data is read from the TCP buffer properly.

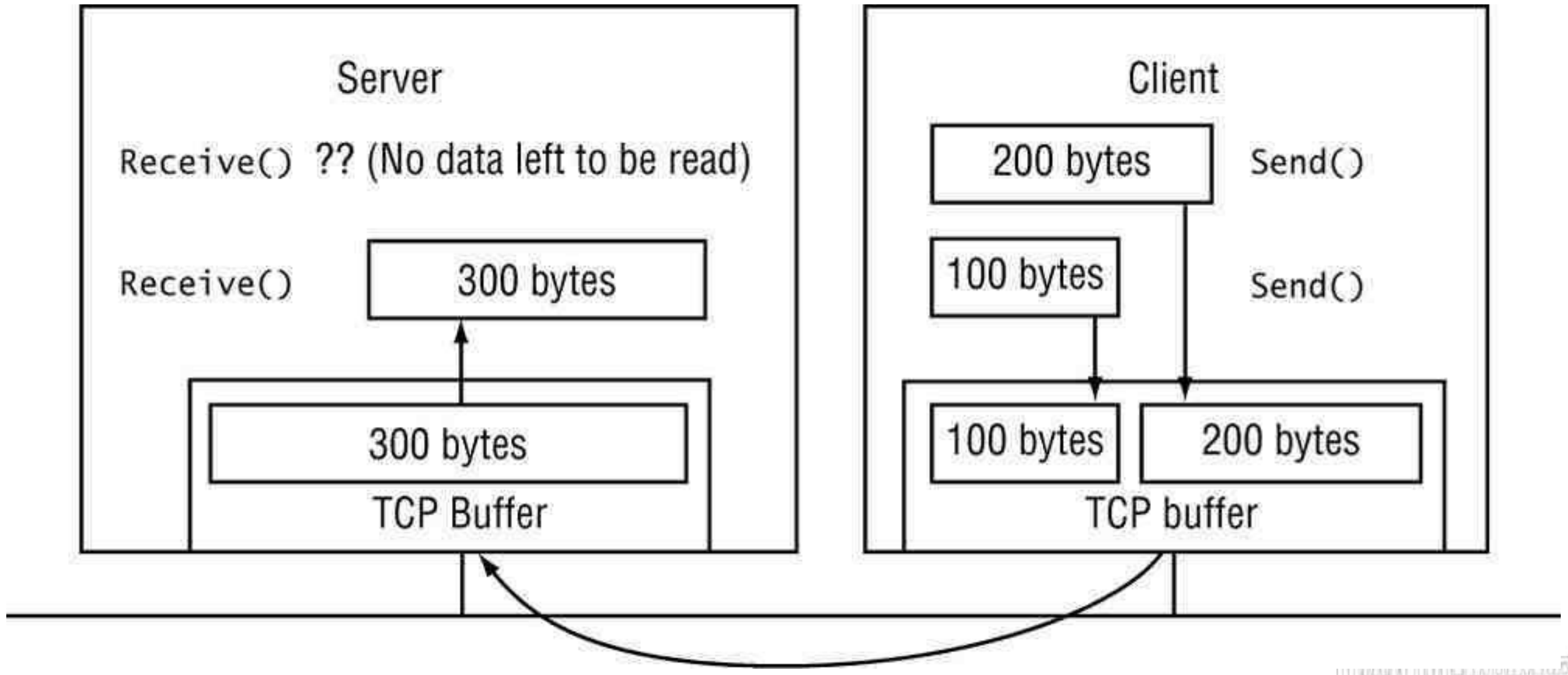
    •Too small a buffer can result in **mismatched messages**

    •Too large a buffer can result in **mixed messages**.



When TCP(Connection-oriented Data Transfer) Goes Bad

- Improper message handling across the network
  - TCP does **not respect message boundaries.**
    •You are not guaranteed that the data from each individual Send() will be read by each individual Receive().



Solving the TCP Message Problem

- “Not realizing the inherent **disregard of message boundaries**, the programmer writes a series of Send() methods on one side, along with a corresponding series of Receive() methods on the other side.”

- **Common ways to distinguish messages sent via TCP:**

  1. Always send **fixed-sized messages**
  2. Use **a marker** system to separate messages
  3. **Send the message size** with each message

Solving the TCP Message Problem

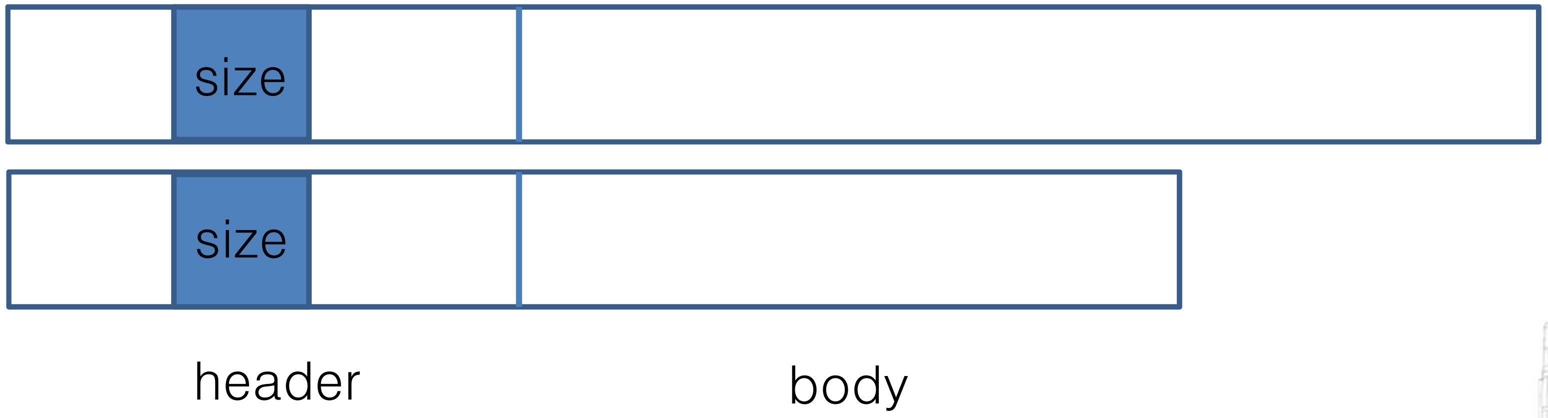
- Always send **fixed-sized messages**
  - create a protocol that always transmits messages of fixed size
  - By setting all messages to the same size, the receiving TCP program can know when the entire message has been received from the remote device

Solving the TCP Message Problem

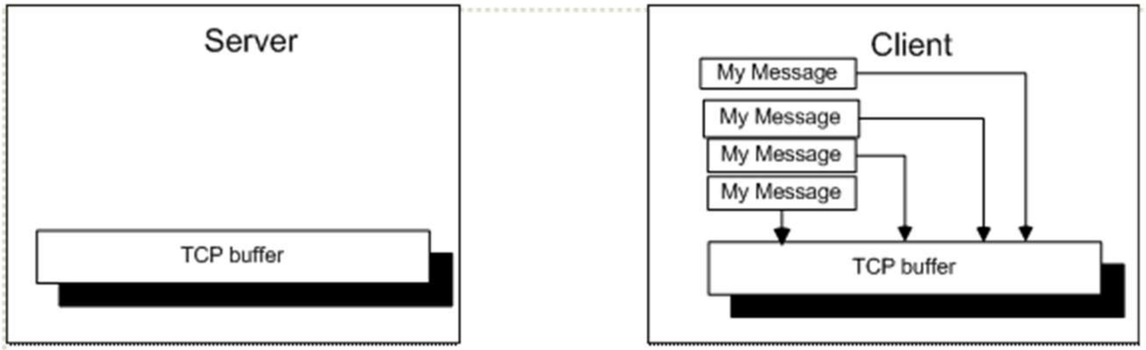
- Use **a marker** system to separate messages
  - separates each message by **a terminating character** to specify the end of the message
  - As messages are received from the socket, the data is checked **character by character for the occurrence of the marker character**.

Solving the TCP Message Problem

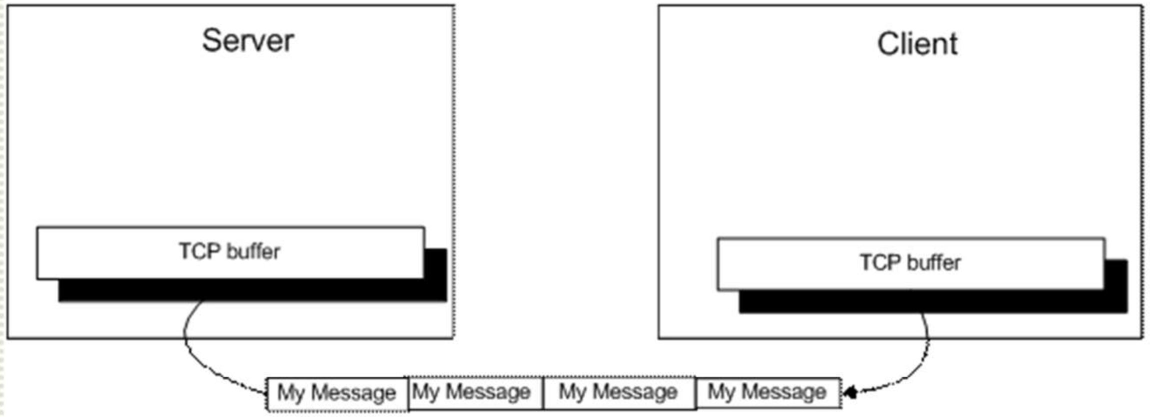
- **Send the message size** with each message
  - create a text representation of the message size(i.e., header) and append it to the beginning of the message
    - read the **message size(header) first**, and then
    - knows how many bytes to read for the complete message



Solving the TCP Message Problem

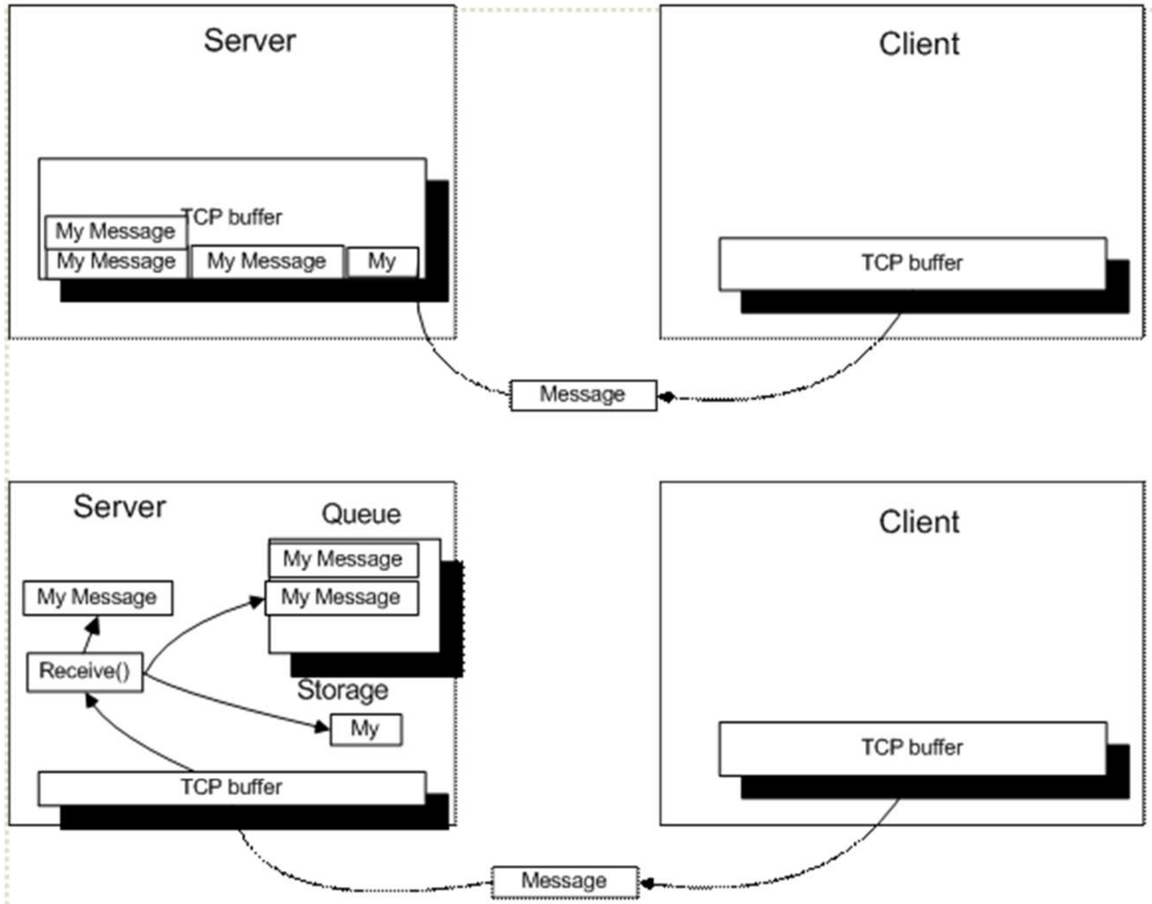


The client program(sender) issues the number of messages and places them into the TCP buffer to be sent to the server (receiver).



The TCP driver sends all of them one after another to the target machine.

Solving the TCP Message Problem

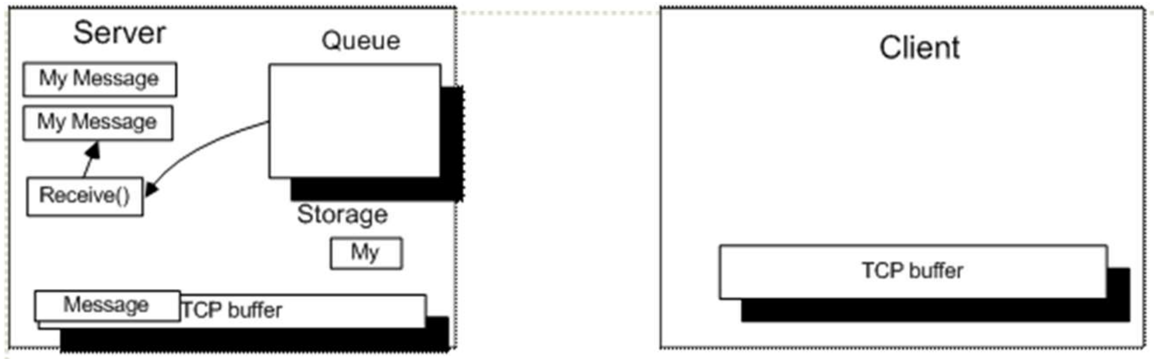


The server's driver receives some of them and places them into the system buffer.

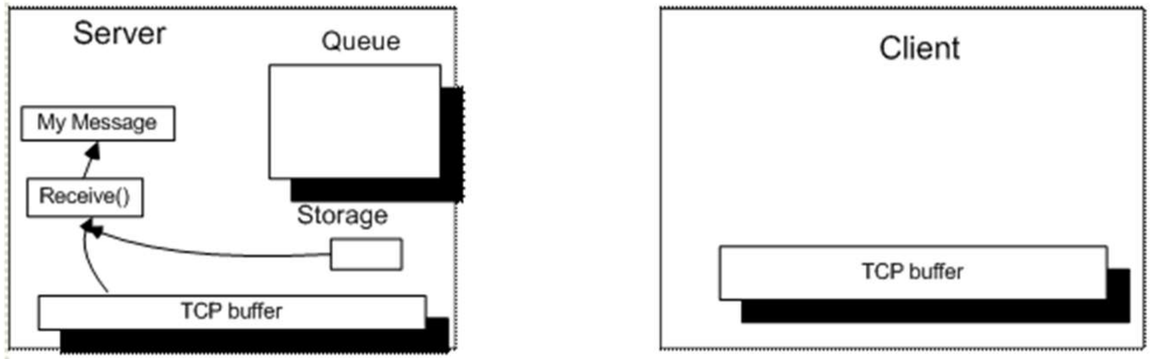
The program reads all the available information from the system buffer and parses it. Every message has a header with a length field so it can know exactly where the message begins and where it ends.

All the messages are placed into a **queue**. If the parser finds that the system buffer holds partial messages, it places them in a special storage.
Finally it returns to program the first message in the raw.

Solving the TCP Message Problem



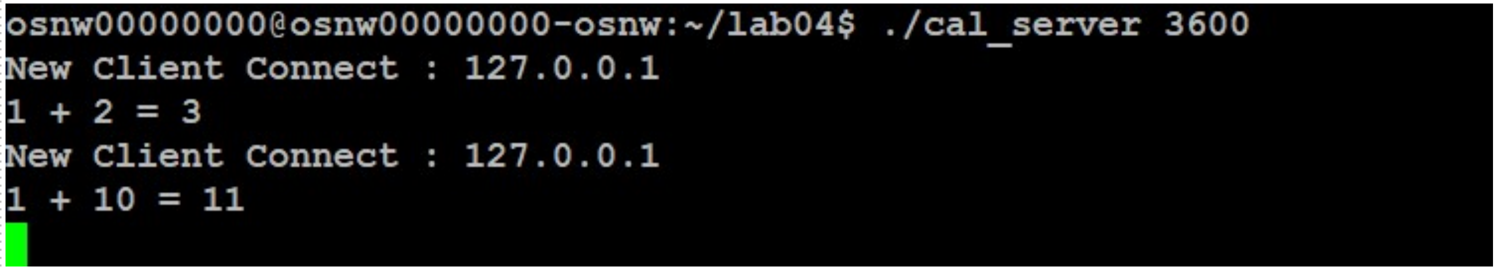
Every time when the program calls Receive(), it checks the availability of messages in a queue and returns one.



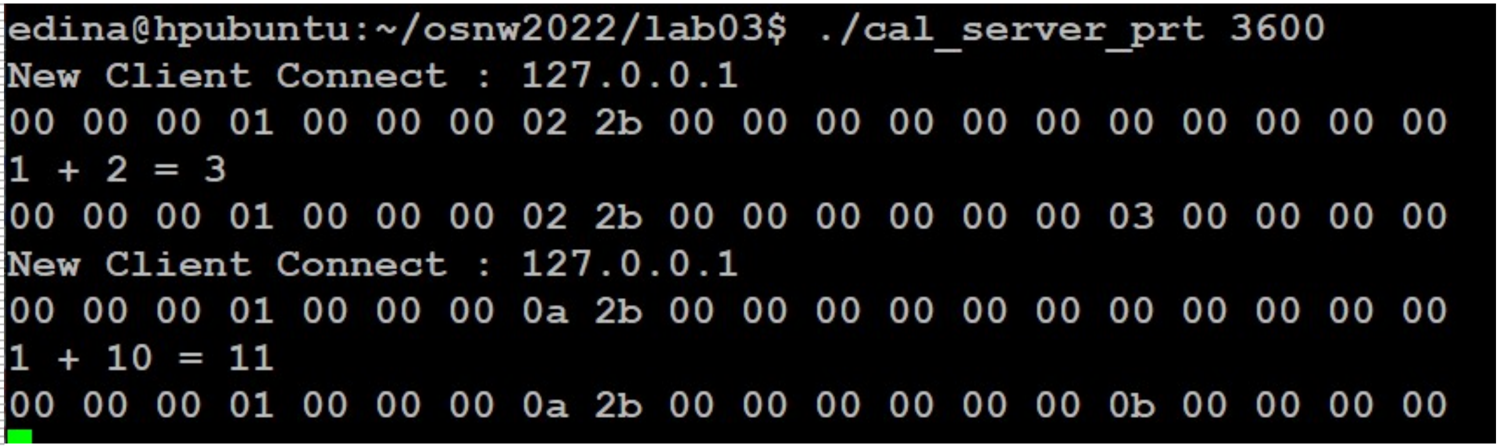
Once the queue is empty, the program reads the system buffer and receives all the information it could accommodate the last time.

cal_server.c / cal_client.c

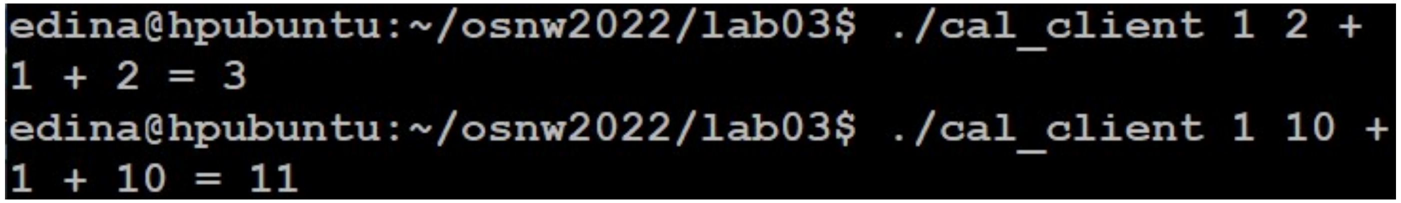
Lab



```
osnw00000000@osnw00000000-osnw:~/lab04$ ./cal_server 3600
New Client Connect : 127.0.0.1
1 + 2 = 3
New Client Connect : 127.0.0.1
1 + 10 = 11
```



```
edina@hpubuntu:~/osnw2022/lab03$ ./cal_server_prt 3600
New Client Connect : 127.0.0.1
00 00 00 01 00 00 00 02 2b 00 00 00 00 00 00 00 00 00 00 00
1 + 2 = 3
00 00 00 01 00 00 00 02 2b 00 00 00 00 00 00 03 00 00 00 00
New Client Connect : 127.0.0.1
00 00 00 01 00 00 00 0a 2b 00 00 00 00 00 00 00 00 00 00 00
1 + 10 = 11
00 00 00 01 00 00 00 0a 2b 00 00 00 00 00 00 0b 00 00 00 00
```



```
edina@hpubuntu:~/osnw2022/lab03$ ./cal_client 1 2 +
1 + 2 = 3
edina@hpubuntu:~/osnw2022/lab03$ ./cal_client 1 10 +
1 + 10 = 11
```

cal_server.c - Linux

```c
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

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

#define PORT 3600

struct cal_data
{
        int left_num;
        int right_num;
        char op;
        int result;
        short int error;
};
```

cal_server.c - Linux

```
int main(int argc, char **argv)
{
        struct sockaddr_in client_addr, sock_addr;
        int listen_sockfd, client_sockfd;
        int addr_len;
        struct cal_data rdata;
        int left_num, right_num, cal_result;
        short int cal_error;

        if( (listen_sockfd  = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        {
                perror("Error ");
                return 1;
        }
        memset((void *)&sock_addr, 0x00, sizeof(sock_addr));
        sock_addr.sin_family = AF_INET;
        sock_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        sock_addr.sin_port = htons(PORT);

        if( bind(listen_sockfd, (struct sockaddr *)&sock_addr, sizeof(sock_addr)) == -1)
        {
                perror("Error ");
                return 1;
        }
```

cal_server.c - Linux

```
if( listen(listen_sockfd, 5) == -1)
{
        perror("Error ");
        return 1;
}

for(;;)
{
        addr_len = sizeof(client_addr);
        client_sockfd = accept(listen_sockfd, (struct sockaddr *)&client_addr, &addr_len);
        if(client_sockfd == -1)
        {
                perror("Error ");
                return 1;
        }
        read(client_sockfd, (void *)&rdata, sizeof(rdata) );

        rdata.error = 0;

        left_num = ntohs(rdata.left_num);
        right_num = ntohs(rdata.right_num);
```

cal_server.c - Linux

```
        switch(rdata.op)
        {
                case '+':   cal_result = left_num + right_num;   break;
                case '-':   cal_result = left_num  - right_num;   break;
                case ‘x':    cal_result = left_num * right_num;    break;
                case '/':    if(right_num == 0)
                             {
                                  cal_error = 2;
                                  break;
                             }
                             cal_result = left_num / right_num;  break;
                default:     cal_error = 1;
        }
        rdata.result = htonl(cal_result);
        rdata.error = htons(cal_error);
        write(client_sockfd, (void *)&rdata, sizeof(rdata));
        close(client_sockfd);
    }

    close(listen_sockfd);
    return 0;
}
```

cal_client.c – Windows Version

```
#include <winsock2.h>
#include <stdio.h>

#define PORT 3600
#define IP "127.0.0.1"

struct cal_data
{
    int left_num;
    int right_num;
    char op;
    int result;
    short int error;
};

int main(int argc, char **argv)
{
    WSADATA WSAData;
    SOCKADDR_IN addr;
    SOCKET s;
    int len;
    int sbyte, rbyte;
    struct cal_data sdata;
```

cal_client.c – Windows Version

```
if (argc != 4)
{
        printf("Usage : %s [num1] [num2] [op]\n", argv[0]);
        return 1;
}

ZeroMemory((void *)&sdata, sizeof(sdata));
sdata.left_num = atoi(argv[1]);
sdata.right_num = atoi(argv[2]);
sdata.op = argv[3][0];

if (WSAStartup(MAKEWORD(2,2), &WSAData) != 0)
{
        return 1;
}

s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET)
{
        return 1;
}
```

cal_client.c – Windows Version

```
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.S_un.S_addr = inet_addr(IP);

if ( connect(s, (struct sockaddr *)&addr, sizeof(addr)) == SOCKET_ERROR)
{
        printf("fail to connect\n");
        closesocket(s);
        return 1;
}

len = sizeof(sdata);
sdata.left_num = htonl(sdata.left_num);
sdata.right_num = htonl(sdata.right_num);
sbyte = send(s, (char *)&sdata, len, 0);
if(sbyte != len)
{
        return 1;
}
```

cal_client.c – Windows Version

```
    rbyte = recv(s, (char *)&sdata, len, 0);
    if(rbyte != len)
    {
            return 1;
    }
    if (ntohs(sdata.error != 0))
    {
            printf("CALC Error %d\n", ntohs(sdata.error));
    }
    printf("%d %c %d = %d\n", ntohl(sdata.left_num), sdata.op,
                                             ntohl(sdata.right_num), ntohl(sdata.result));

    closesocket(s);
    WSACleanup( );
    return 0;
}
```

Thank You !

뇌를 자극하는 TCP/IP 소켓 프로그래밍