

# 컴퓨터 네트워크 Client-Server Program

201811420 컴퓨터공학전공 김동건

## 목차

1. 프로그램 동작 구조 및 소스코드 분석.....	2
1.1 TCP Protocol 동작 구조.....	2
1.2 TCP Protocol 소스코드 분석.....	2
1.3 UDP Protocol 동작 구조.....	8
1.4 UDP Protocol 소스코드 분석.....	8
2. 프로그램 동작 원리.....	11
2.1 데이터 전송 Protocol 정의.....	12
3. 통신 과정 분석.....	11
3.1 TCP Wireshark Packet 분석.....	12
3.2 UDP Wireshark Packet 분석.....	15
4. 소스 코드 및 참고 자료.....	15
4.1 소스 코드.....	17
4.2 참고 자료.....	17

## 그림 목차

그림 1: TCP 프로그램 전체 동작 구조.....	3
그림 2: UDP 프로그램 전체 동작 구조.....	8
그림 3: TCP 3단계 연결 Flow Graph.....	12
그림 4: TCP 3단계 연결 WireShark.....	13
그림 5: TCP Client 수식 전송 Flow Graph.....	13
그림 6: Client 수식정보 전송 WireShark.....	13
그림 7: Client Data Packet Payload.....	13
그림 8: TCP Client 결과값 Flow Graph.....	13
그림 9: TCP Client의 결과값 WireShark.....	13
그림 10: Client Response Packet Payload.....	14
그림 11: AddServer와 통신 FlowGraph.....	14
그림 12: AddSubTcpServer와 통신 Packet Payload.....	15
그림 13: AddSubTcpServer와 통신 계산결과 Packet Payload.....	15
그림 14: AddSubTcpServer와 TCP 3단계 연결 종료.....	15
그림 15: 모든 UDP Packet.....	15
그림 16: ClientUdp와 InputParseUdpServer 수식 전송 Packet.....	16
그림 17: InputParseUdpServer의 결과값 응답 Packet.....	16
그림 18: UDP 정의된 프로토콜로 수식 전송 Packet.....	16

# 1. 프로그램 동작 구조 및 소스코드 분석

## 1.1 TCP Protocol 동작 구조

TCP Protocol로 구현한 프로그램의 전체 동작 구조는 그림1과 같다.

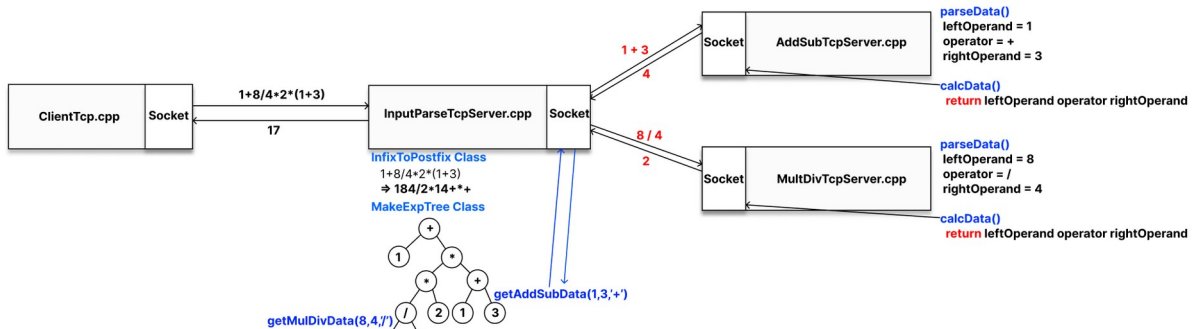


그림 1: TCP 프로그램 전체 동작 구조

ClientTcp.cpp에서 Tcp Socket 객체 생성 및 사용자의 입력을 받고 이를 InputParseTcpServer로 그대로 전송한다. InputParseTcpServer에서는 InfixToPostfix class를 통해 Socket으로 전달받은 값을 후위표기식으로 변환하고 MakeExpTree Class를 통해 후위표기식 트리를 만든다. MakeExpTree Class의 calc 메서드를 호출해 값을 계산한다. calc메서드는 다시 getAddSubData() 또는 getMulDivData()메서드를 호출한다. 이 메서드들은 AddSubTcpServer 또는 MultDivTcpServer로 “leftOperand Operator rightOperand” 형식으로 전송한다. 각각의 TcpServer는 parseData()메서드를 통해 분리하고 값을 계산하여 다시 InputParseTcpServer로 전송한다. 최종적으로 다시 ClientTcp로 전송하여 전체 계산을 완료한다.

## 1.2 TCP Protocol 소스코드 분석

### 1.2.1 ClientTcp.cpp

```
int main() {
    auto socketModule = make_unique<ClientSocket>();
    string ipAddress = "127.0.0.1";
    if(socketModule->socketConnect(ipAddress)) {
        cout << "Input expression" << endl;
        string inputExpression;
        cin >> inputExpression;
        const string&& result = socketModule->sendPayloadReceiveData(inputExpression);
        cout << "Result : " << result << endl;
    } else {
        cout << "Failed to Connect" << endl;
    }
    return 0;
}
```

Main 함수에서는 사용자의 입력을 받고 서버로 전송 및 결과를 받는다.

```

bool socketConnect(const string& ipAddress) {
    if ((sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) {
        perror("socket");
        return false;
    }
    their_addr.sin_family = AF_INET; /* host byte order */
    their_addr.sin_port = htons(port); /* short, network byte order */
    // their_addr.sin_addr = *((struct in_addr *)he->h_addr);
    their_addr.sin_addr.s_addr = inet_addr(ipAddress.c_str());
    bzero(&(their_addr.sin_zero), 8); /* zero the rest of the struct */

    if (connect(sockfd, (struct sockaddr*)&their_addr, sizeof(struct sockaddr)) == -1) {
        perror("connect");
        return false;
    }
    return true;
}

```

SocketConnect() 메서드에서는 TCP 3단계 연결 후 true, false를 반환한다.

```

string sendPayloadReceiveData(const string& payload) {
    if (send(sockfd, payload.c_str(), payload.size() / sizeof(char), 0) == -1) {
        perror("send");
        exit(1);
    }
    char buf[maxDataSize];
    if ((numbytes = recv(sockfd, buf, maxDataSize, 0)) == -1) {
        perror("recv");
        exit(1);
    }
    buf[numbytes] = '\0';
    if (close(sockfd) == -1) {
        perror("close");
        exit(1);
    }
    return string(buf);
}

```

SendPayloadReceiveData() 메서드에서는 TCP 3단계 연결 성공 후 서버로 사용자의 입력을 전송하고 recv() 함수를 통해 데이터를 받은 후 결과를 string 형식으로 리턴한다.

### 1.2.2 InputParseTcpServer.cpp / MultDivTcpServer.cpp

```

int main() {
    auto listenTcpServer = make_unique<ListenTCPServer>();
    auto makeTreeModule = make_shared<MakeExpTree>();
    listenTcpServer->startListen(makeTreeModule);
    return 0;
}

```

main 함수에서는 ListenTCPServer를 시작하고 후위표기식 트리클래스를 만들어 서버 클래스로 전달한다.

```

void startListen(auto makeTreeModule) {
    if ((socketFd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1) { // 1. Create TCP Socket
        perror("socket");
        exit(1);
    }

    myAddr.sin_family = AF_INET; /* host byte order */
    myAddr.sin_port = htons(listenInputParsePort); /* short, network byte order */ // 2. Set Server Port 3490
    myAddr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */ // 3. Set Server IP Address
    bzero(&(myAddr.sin_zero), 8); /* zero the rest of the struct */

    socklen_t reuseAddress = 1;
    if (setsockopt(socketFd, SOL_SOCKET, SO_REUSEADDR, (const char *)&reuseAddress, sizeof(reuseAddress)) == -1) {
        perror("setsockopt"); // Port Reuse Option!
    }

    if (bind(socketFd, (struct sockaddr *)&myAddr, sizeof(struct sockaddr)) == -1) { // 4. Bind IP, Port
        perror("bind");
        exit(1);
    }

    if (listen(socketFd, backlog) == -1) { // 5. Listen! => TCP 3단계 Connection
        perror("listen");
        exit(1);
    }

    while (1) { /* main accept() loop */
        sin_size = sizeof(struct sockaddr_in);
        if ((newFd = accept(socketFd, (struct sockaddr *)&theirAddr,
                           &sin_size)) == -1) { // 6. Accept!
            perror("accept");
            continue;
        }
        cout << "server: got connection from " << inet_ntoa(theirAddr.sin_addr) << endl;
        if (!fork()) { /* this is the child process */ // 7. then, Child Process do Any Jobs..
            cout << getpid() << endl;
            char buf[maxDataSize];
            if ((numbytes = recv(newFd, buf, maxDataSize, 0)) == -1) {
                perror("recv");
                exit(1);
            }
            buf[numbytes] = '\0';
            cout << "Input: " << buf << endl;
            string userInput(buf);
            auto infixToPostModule = make_unique<InfixToPostfix>();
            const vector<string> &tmp = infixToPostModule->infixToPostfix(userInput);
            const Node *postRoot = makeTreeModule->makeExpTree(tmp);
            string result = to_string(makeTreeModule->calc(postRoot));

            if (send(newFd, result.c_str(), result.size(), 0) == -1)
                perror("send");
            cout << "Connction Successful! Message! :" << buf << endl;
            close(newFd);
            exit(0);
        }
        close(newFd); /* parent doesn't need this */
    }
}

```

TCP listen() 함수를 호출해 입력을 대기후 recv() 함수로 실제 입력을 받은후 InfixToPostfix 클래스를통해 후위표기식으로 바꾼후 MakeExpTree Class의 calc함수를 호출해 실제 계산을 실행후 결과를 Client로 전송한다.



```

float calc(const Node *const root) {
    if (root->mleft == nullptr && root->mright == nullptr)
        return stof(root->mexp);
    float leftOperand = calc(root->mleft);
    float rightOperand = calc(root->mright);
    switch (static_cast<char>(stoi(root->mexp))) {
        case '+':
            return ListenTCPServer::getAddSubData(leftOperand, rightOperand, '+');
        case '-':
            return ListenTCPServer::getAddSubData(leftOperand, rightOperand, '-');
        case '*':
            return ListenTCPServer::getMulDivData(leftOperand, rightOperand, '*');
        case '/':
            return ListenTCPServer::getMulDivData(leftOperand, rightOperand, '/');
        default:
            cout << "What is This ?!?" << root->mexp << endl;
            exit(1);
    }
}

```

MakeExpTree::clac() 메서드는 AddSubTcpServer, MultDivTcpServer를 직접적으로 호출하는 메서드인 getAddSubData(), getMulDivData()를 호출한다. 다음 두 메서드의

```

string getData(const int &port, const char *payload, int size) {
    struct sockaddr_in their_addr; /* connector's address information */
    if ((socketFd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }
    their_addr.sin_family = AF_INET; /* host byte order */
    their_addr.sin_port = htons(port); /* short, network byte order */
    their_addr.sin_addr.s_addr = inet_addr(serverAddress);
    bzero(&(their_addr.sin_zero), 8); /* zero the rest of the struct */

    if (connect(socketFd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1) {
        perror("connect");
        exit(1);
    }
    if (send(socketFd, payload, size, 0) == -1) {
        perror("send");
    }
    char buf[maxDataSize];
    if ((numbytes = recv(socketFd, buf, maxDataSize, 0)) == -1) {
        perror("recv");
        exit(1);
    }
    buf[numbytes] = '\0';
    close(socketFd);
    return string(buf);
}

```

내부는 getData() 메서드로 구성되어있고 getData()는 실제로 TCP 통신을 수행한다.

### 1.2.3 AddSubTcpServer.cpp

```
int main() {  
    auto test = make_unique<ListenTCPSever>();  
    test->startListen();  
    return 0;  
}
```

main함수에서는 단순히 ListenTCPSever 클래스의 startListen() 메서드를 호출한다.

```
while (1) { /* main accept() loop */  
    sin_size = sizeof(struct sockaddr_in);  
    if ((newFd = accept(socketFd, (struct sockaddr *)&theirAddr,  
                        &sin_size)) == -1) { // 6. Accept!  
        perror("accept");  
        continue;  
    }  
    cout << "server: got connection from " << inet_ntoa(theirAddr.sin_addr) << endl;  
    if (!fork()) { /* this is the child process */ // 7. then, Child Process do Any  
        char buf[maxDataSize];  
        if ((numbytes = recv(newFd, buf, maxDataSize, 0)) == -1) {  
            perror("recv");  
            exit(1);  
        }  
        buf[numbytes] = '\0';  
        parseData(string(buf));  
        const string& result = string(calcData());  
        if (send(newFd, result.c_str(), result.size(), 0) == -1)  
            perror("send");  
        close(newFd);  
        exit(0);  
    }  
    close(newFd); /* parent doesn't need this */  
    while (waitpid(-1, NULL, WNOHANG) > 0)  
        ; /* clean up child processes */  
}
```

StartListen() 메서드에서는 TCP 3단계 연결을 accept하고 데이터를 recv() 함수를 통해 받은후 parseData() 메서드를 통해 입력을 leftOperand, operator, rightOperand로 구별한다.

```

string calcData() {
    switch(get<1>(expTuple)){
        case '+' :
            return to_string(get<0>(expTuple) + get<2>(expTuple));
        case '-' :
            return to_string(get<0>(expTuple) - get<2>(expTuple));
        default :
            cout<< "Unknown Error!" <<endl;
            exit(1);
    }
}

```

그후 calcData() 메서드를 호출해 계산한 결과를 전송한다.

### 1.3 UDP Protocol 동작 구조

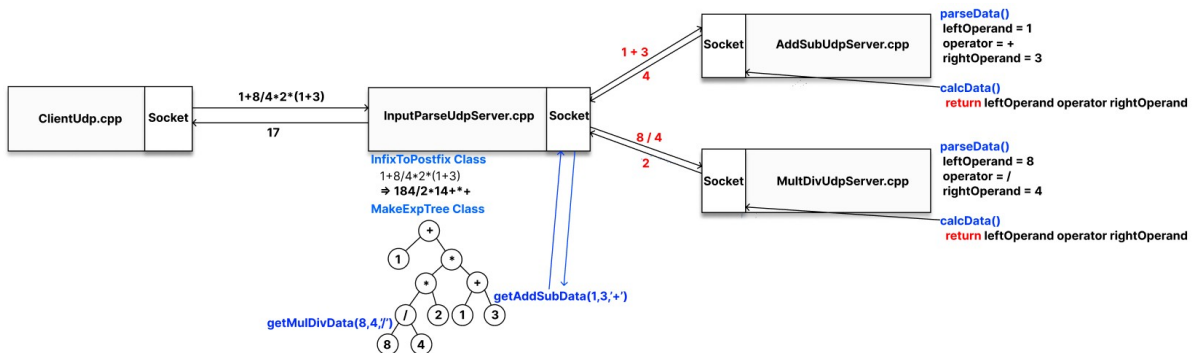


그림 2: UDP 프로그램 전체 동작 구조

UDP 프로그램 또한 ClientUdp.cpp에서 UDP Socket 객체 생성 및 사용자의 입력을 받고 이를 InputParseUdpServer로 그대로 전송한다. InputParseUdpServer에서는 InfixToPostfix class를 통해 Socket으로 전달받은 값을 후위표기식으로 변환하고 MakeExpTree Class를 통해 후위표기식 트리를 만든다. MakeExpTree Class의 calc 메서드를 호출해 값을 계산한다. calc 메서드는 다시 getAddSubData() 또는 getMultDivData()메서드를 호출한다. 이 메서드들은 AddSubUdpServer 또는 MultDivUdpServer로 “leftOperand Operator rightOperand” 형식으로 전송한다. 각각의 UdpServer는 parseData()메서드를 통해 분리하고 값을 계산하여 다시 InputParseUdpServer로 전송한다. 최종적으로 다시 ClientUdp로 전송하여 전체 계산을 완료한다.

### 1.4 UDP Protocol 소스코드 분석

#### 1.4.1 ClientUdp.cpp

```

int main() {
    auto socketModule = make_unique<ClientUdpSocket>();
    string ipAddress = "127.0.0.1";
    cout << "Input expression" << endl;
    string inputExpression;
    cin >> inputExpression;
    const string&& result = socketModule->sendPayloadReceiveData(ipAddress, inputExpression);
    cout << "Result : " << result << endl;
    return 0;
}

```

ClientTcp.cpp와는 다르게 3단계 연결 과정이 없으므로 사용자의 입력을 바로 sendPayloadReceiveData() 메서드에 전달한다.

```

string sendPayloadReceiveData(const string& ipAddress, const string& payload) {
    if ((socketfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
        perror("socket");
        exit(1);
    }
    their_addr.sin_family = AF_INET; /* host byte order */
    their_addr.sin_port = htons(port); /* short, network byte order */
    their_addr.sin_addr.s_addr = inet_addr(ipAddress.c_str());
    bzero(&(their_addr.sin_zero), 8); /* zero the rest of the struct */

    if (sendto(socketfd, payload.c_str(), payload.size(), 0, (struct sockaddr*)&their_addr, sizeof(struct sockaddr)) == -1) {
        perror("sendto");
        exit(1);
    }
    char buf[maxDataSize];
    if ((numbytes = recv(socketfd, buf, maxDataSize, 0)) == -1) {
        perror("recv");
        exit(1);
    }
    buf[numbytes] = '\0';
    if (close(socketfd) == -1) {
        perror("close");
        exit(1);
    }
    return string(buf);
}

```

UDP Socket 객체를 만들고 sendto() 함수를 통해 값을 전송후 recv() 함수로 계산결과를 받고 출력한다.

#### 1.4.2 InputParseUdpServer.cpp

```

int main() {
    auto listenUdpServer = make_unique<ListenUDPServer>();
    auto makeTreeModule = make_shared<MakeExpTree>();
    listenUdpServer->startListen(makeTreeModule);
    return 0;
}

```

main함수에서는 ListenUDPServer 클래스를 생성하고 후위표기식 트리를 만드는 객체를 전달한다.

```

void startListen(auto makeTreeModule) {
    while (1) { /* main accept() loop */
        if ((socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) { // 1. Create TCP Socket
            perror("socket");
            exit(1);
        }
        myAddr.sin_family = AF_INET; /* host byte order */
        myAddr.sin_port = htons(listenInputParsePort); /* short, network byte order */ // 2. Set Server Port 3490
        myAddr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */ // 3. Set Server IP Address
        bzero(&(myAddr.sin_zero), 8); /* zero the rest of the struct */
        socklen_t reuseAddress = 1;
        if (setsockopt(socketFd, SOL_SOCKET, SO_REUSEADDR, (const char*)&reuseAddress, sizeof(reuseAddress)) == -1) {
            perror("setsockopt"); // Port Reuse Option!
        }
        if (bind(socketFd, (struct sockaddr*)&myAddr, sizeof(struct sockaddr)) == -1) { // 4. Bind IP, Port
            perror("bind");
            exit(1);
        }
        socklen_t addr_len = sizeof(struct sockaddr);
        if ((numbytes = recvfrom(socketFd, buf, maxDataSize, 0, (struct sockaddr*)&their_addr, &addr_len)) == -1) {
            perror("recvfrom");
            exit(1);
        }
        buf[numbytes] = '\0';
    }
}

```

Recvfrom() 함수를 통해 UDP Datagram을 받는다.



```

string userInput(buf);
auto infixToPostModule = make_unique<InfixToPostfix>();
const vector<string> &tmp = infixToPostModule->infixToPostfix(userInput);
const Node *postRoot = makeTreeModule->makeExpTree(tmp);
string result = to_string(makeTreeModule->calc(postRoot));
//string result = "test";
if (sendto(socketFd, result.c_str(), result.size(), 0, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1) {
    perror("send");
}
cout << "Connction Successful! Message! :" << buf << endl;
close(socketFd);

```

Calc() 메서드를 통해 각각 구현한 서버로 계산을 실행하며 결과값을 sendto() 함수로 전송한다.

```

float calc(const Node *const root) {
    if (root->mleft == nullptr && root->mright == nullptr)
        return stof(root->mexp);
    float leftOperand = calc(root->mleft);
    float rightOperand = calc(root->mright);
    switch (static_cast<char>(stoi(root->mexp))) {
        case '+':
            return ListenUDPServer::getAddSubData(leftOperand, rightOperand, '+');
        case '-':
            return ListenUDPServer::getAddSubData(leftOperand, rightOperand, '-');
        case '*':
            return ListenUDPServer::getMulDivData(leftOperand, rightOperand, '*');
        case '/':
            return ListenUDPServer::getMulDivData(leftOperand, rightOperand, '/');
        default:
            cout << "What is This ?!?" << root->mexp << endl;
            exit(1);
    }
}

```

calc() 메서드에서는 leftOperand와 rightOperand를 각 UDP Server로 전달한다.

```

string getData(const int &port, const char *payload, int size) {
    if ((socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
        perror("socket");
        exit(1);
    }
    their_addr.sin_family = AF_INET; /* host byte order */
    their_addr.sin_port = htons(port); /* short, network byte order */
    their_addr.sin_addr.s_addr = inet_addr(serverAddress);
    bzero(&(their_addr.sin_zero), 8); /* zero the rest of the struct */

    if (sendto(socketFd, payload, size, 0, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1) {
        perror("sendto");
        exit(1);
    }
    char buf[maxDataSize];
    if ((numbytes = recv(socketFd, buf, maxDataSize, 0)) == -1) {
        perror("recv");
        exit(1);
    }
    buf[numbytes] = '\0';
    if (close(socketFd) == -1) {
        perror("close");
        exit(1);
    }
    return string(buf);
}

```

실제 값을 전송하는 getData() 메서드에서 sendto() 함수와 recv() 함수를 통해서 값을 받는다.

### 1.4.3 AddSubUdpServer.cpp / MultDivUdpServer.cpp

```
int main() {
    auto listenUdpServer = make_unique<ListenUdpServer>();
    listenUdpServer->startListen();
    return 0;
}
```

main() 함수에서는 서버를 실행시킨다.

```
if ((socketFd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) { // 1. Create TCP Socket
    perror("socket");
    exit(1);
}
myAddr.sin_family = AF_INET; /* host byte order */
myAddr.sin_port = htons(sendAddSubPort); /* short, network byte order */ // 2. Set Server Port 3490
myAddr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */ // 3. Set Server IP Address
bzero(&(myAddr.sin_zero), 8); /* zero the rest of the struct
char buf[maxDataSize];
socklen_t reuseAddress = 1;
if (setsockopt(socketFd, SOL_SOCKET, SO_REUSEADDR, (const char *)&reuseAddress, sizeof(reuseAddress)) == -1) {
    perror("setsockopt"); // Port Reuse Option!
}

if (bind(socketFd, (struct sockaddr *)&myAddr, sizeof(struct sockaddr)) == -1) { // 4. Bind IP, Port
    perror("bind");
    exit(1);
}
socklen_t addr_len = sizeof(struct sockaddr);
if ((numbytes = recvfrom(socketFd, buf, maxDataSize, 0,
    (struct sockaddr *)&their_addr, &addr_len)) == -1) {
    perror("recvfrom");
    exit(1);
}
buf[numbytes] = '\0';
parseData(string(buf));
string&& result = calcData();
```

startListen() 메서드에서는 UDP Socket 객체를 생성하고 recvfrom() 함수로 값을 받고 parseData(), calcData() 메서드를 통해 값을 파싱 및 계산한다.

```
if (sendto(socketFd, result.c_str(), result.size(), 0, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1) {
    perror("send");
}
cout << "Connction Successful! Message! :" << buf << endl;
close(socketFd);
```

계산결과를 sendto() 함수를 통해 전송한다.

```
string calcData() {
    switch (get<1>(expTuple)) {
        case '+':
            return to_string(get<0>(expTuple) + get<2>(expTuple));
        case '-':
            return to_string(get<0>(expTuple) - get<2>(expTuple));
        default:
            cout << "Unknown Error!" << endl;
            exit(1);
    }
}
```

CalcData() 메서드는 실제 값을 계산한다

## 2. 프로그램 동작 원리

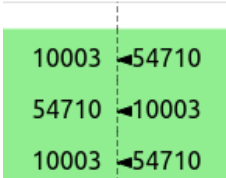
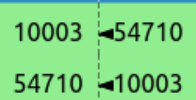
### 2.1 데이터 전송 Protocol 정의

구분	InputParseServer	
	Data Format	Port
Client	수식 <b>String</b>	UDP: 20000 TCP: 10003
AddSubServer	(\d*\.\d*\d+)\s\+ \-\s(\d*\.\d*\d+) (\d*\.\d*\d+)\s\+ \-\s(\d*\.\d*\d+)	UDP: 20001 TCP: 10001
MultDivServer	(\d*\.\d*\d+)\s\* \/\s(\d*\.\d*\d+) (\d*\.\d*\d+)\s\* \/\s(\d*\.\d*\d+)	UDP: 20002 TCP: 10002

표 1: 데이터 전송 TCP Protocol

## 3. 통신 과정 분석

### 3.1 TCP Wireshark Packet 분석

3.1.1 ClientTcp와 InputParseTcpServer					
1단계 TCP 3단계 연결					
					
그림 3: TCP 3단계 연결 Flow Graph					
2	0.396109327	127.0.0.1	1번	127.0.0.1	TCP 76 54710 → 10003 [SYN] Seq=
3	0.396117792	127.0.0.1	2번	127.0.0.1	TCP 76 10003 → 54710 [SYN, ACK]
4	0.396124496	127.0.0.1	3번	127.0.0.1	TCP 68 54710 → 10003 [ACK] Seq=
그림 4: TCP 3단계 연결 Wireshark					
1단계 → 1번 Client는 운영체제로부터 자동으로 할당된 54710 포트에서 서버로 SYN를 전송					
1단계 → 2번 서버는 Client의 SYN신호를 보고 SYN, ACK로 응답					
1단계 → 3번 Client는 SYN, ACK를 응답받고 ACK 응답후 TCP 3단계 연결 성공					
2단계 Client 수식 전송					
					
그림 5: TCP Client 수식 전송 Flow Graph					
8	1.252403077	127.0.0.1	1번	127.0.0.1	TCP 93 54710 → 10003 [PSH, ACK] Seq=
9	1.252422840	127.0.0.1	2번	127.0.0.1	TCP 68 10003 → 54710 [ACK] Seq=1
그림 6: Client 수식정보 전송 Wireshark					
2단계 → 1번 Client는 수식 Data를 Server에 전송한다. Packet의 Payload의 모습은 그림과 같다.					

▶ Frame 8: 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 54710, Dst Port: 10003, Seq: 1, Ack: 1, Len: 25  
 ▶ Data (25 bytes)

Data: 28312b32292f322a372d35302b39362d2832372d3330292f35  
 [Length: 25]

0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....
0010	45 00 00 4d 30 04 40 00	40 06 0c a5 7f 00 00 01	E..M0.@. @.....
0020	7f 00 00 01 d5 b6 27 13	1c d4 c0 2e b4 b8 d8 55	.....'.....U
0030	80 18 02 00 fe 41 00 00	01 01 08 0a c2 6f 03 e7	.....A.....o..
0040	c2 6f 00 8f 28 31 2b 32	29 2f 32 2a 37 2d 35 30	.o..(1+2 )/2*7-50
0050	2b 39 36 2d 28 32 37 2d	33 30 29 2f 35	+96-(27- 30)/5

그림 7: Client Data Packet Payload

2단계→ 2번 Server의 ACK 응답

3단계 결과값 응답

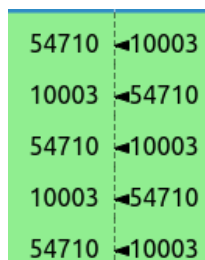


그림 8: TCP Client 결과값 Flow Graph

90	1.258859308	127.0.0.1	1번	127.0.0.1	TCP	77 10003 → 54710 [PSH, ACK]
91	1.258865175	127.0.0.1	2번	127.0.0.1	TCP	68 54710 → 10003 [ACK] Seq=2
92	1.258874120	127.0.0.1	3번	127.0.0.1	TCP	68 10003 → 54710 [FIN, ACK]
93	1.258993915	127.0.0.1	4번	127.0.0.1	TCP	68 54710 → 10003 [FIN, ACK]
94	1.259010565	127.0.0.1	5번	127.0.0.1	TCP	68 10003 → 54710 [ACK] Seq=2

그림 9: TCP Client의 결과값 WireShark

3단계→ 1번

▶ Frame 90: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 10003, Dst Port: 54710, Seq: 1, Ack: 26, Len: 9  
 ▶ Data (9 bytes)

Data: 35372e303939393938  
 [Length: 9]

0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....
0010	45 00 00 3d b5 78 40 00	40 06 87 40 7f 00 00 01	E..=-x@. @..@....
0020	7f 00 00 01 27 13 d5 b6	b4 b8 d8 55 1c d4 c0 47	....'.....U...G
0030	80 18 02 00 fe 31 00 00	01 01 08 0a c2 6f 03 ed	.....1.....o..
0040	c2 6f 03 e7 35 37 2e 30	39 39 39 39 38	.o..57.0 99998

그림 10: Client Response Packet Payload

이미 Client, Server와 3단계 연결이 되어있으므로 바로 결과 값을 PSH 플래그와 함께 전송한다.

3단계→ 2번 Client의 ACK응답

3단계→ 3번 Server의 종료 플래그 FIN, ACK

3단계→ 4번 Client의 종료 응답 FIN, ACK

3단계→ 5번 Server의 응답 ACK



### 3.1.2 InputParseTcpServer와 AddSubTcpServer / MultDivTcpServer

#### 1단계 TCP 3단계 연결

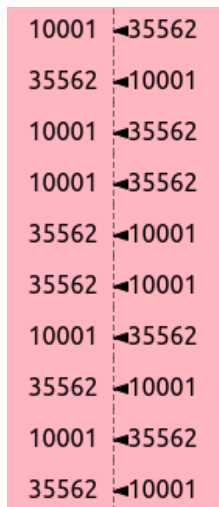


그림 11: AddServer와 통신 FlowGraph

10	1.252745403	127.0.0.1	127.0.0.1	TCP	76	35562 → 10001	[SYN]	Seq=
11	1.252756200	127.0.0.1	127.0.0.1	TCP	76	10001 → 35562	[SYN, ACK]	
12	1.252765660	127.0.0.1	127.0.0.1	TCP	68	35562 → 10001	[ACK]	Seq=

위에서 설명한바와 같이 운영체제에 의해 자동으로 할당된 35565 포트에서 10001번 AddSubTcpServer와 TCP 3단계 연결을 수행한다.

#### 2단계 정의된 프로토콜로 수식 전송

▶ Frame 13: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 35562, Dst Port: 10001, Seq: 1, Ack: 1, Len: 20  
 ▼ Data (20 bytes)  
 Data: 312e30303030303020343320322e303030303030  
 [Length: 20]

0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....
0010	45 00 00 48 c5 6a 40 00	40 06 77 43 7f 00 00 01	E..H.j@. @.wC...
0020	7f 00 00 01 8a ea 27 11	66 1b 5e 17 08 02 ae 23	.....'. f.^...#
0030	80 18 02 00 fe 3c 00 00	01 01 08 0a c2 6f 03 e7	.....<.....o..
0040	c2 6f 03 e7 31 2e 30 30	30 30 30 30 20 34 33 20	..o..1.00 0000 43
0050	32 2e 30 30 30 30 30 30		2.0000000

그림 12: AddSubTcpServer와 통신 Packet Payload

정의된 프로토콜 형식으로 계산식을 전달한다.

#### 3단계 결과값 응답

▶ Frame 15: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0  
 ▶ Linux cooked capture v1  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 10001, Dst Port: 35562, Seq: 1, Ack: 21, Len: 8  
 ▼ Data (8 bytes)  
 Data: 332e303030303030  
 [Length: 8]

0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....
0010	45 00 00 3c 3e 4b 40 00	40 06 fe 6e 7f 00 00 01	E.<>K@. @..n...
0020	7f 00 00 01 27 11 8a ea	08 02 ae 23 66 1b 5e 2b	.....'. ...#f.^+
0030	80 18 02 00 fe 30 00 00	01 01 08 0a c2 6f 03 e8	.....0.....o..
0040	c2 6f 03 e7 33 2e 30 30	30 30 30 30	..o..3.00 0000

그림 13: AddSubTcpServer와 통신 계산결과 Packet Payload

계산 결과를 전송후 TCP 연결을 종료한다.

17	1.253465440	127.0.0.1	127.0.0.1	TCP	68	10001 → 35562	[FIN, ACK]	
18	1.253585926	127.0.0.1	127.0.0.1	TCP	68	35562 → 10001	[FIN, ACK]	
19	1.253607966	127.0.0.1	127.0.0.1	TCP	68	10001 → 35562	[ACK]	Seq=

그림 14: AddSubTcpServer와 TCP 3단계 연결 종료

## 3.2 UDP Wireshark Packet 분석

### 3.2.1 ClientUdp와 InputParseUdpServer

5	1.624851279	127.0.0.1	127.0.0.1	UDP	69 58227 → 20000 Len=25
6	1.625096137	127.0.0.1	127.0.0.1	UDP	64 58208 → 20001 Len=20
7	1.625275633	127.0.0.1	127.0.0.1	UDP	52 20001 → 58208 Len=8
8	1.625401100	127.0.0.1	127.0.0.1	UDP	64 38769 → 20002 Len=20
9	1.625560714	127.0.0.1	127.0.0.1	UDP	52 20002 → 38769 Len=8
10	1.625682562	127.0.0.1	127.0.0.1	UDP	64 36966 → 20002 Len=20
11	1.625808819	127.0.0.1	127.0.0.1	UDP	53 20002 → 36966 Len=9
12	1.625933740	127.0.0.1	127.0.0.1	UDP	66 49456 → 20001 Len=22
13	1.626084065	127.0.0.1	127.0.0.1	UDP	54 20001 → 49456 Len=10
14	1.626203711	127.0.0.1	127.0.0.1	UDP	67 35294 → 20001 Len=23
15	1.626306724	127.0.0.1	127.0.0.1	UDP	53 20001 → 35294 Len=9
16	1.626426599	127.0.0.1	127.0.0.1	UDP	66 52192 → 20001 Len=22
17	1.626536270	127.0.0.1	127.0.0.1	UDP	53 20001 → 52192 Len=9
18	1.626571579	127.0.0.1	127.0.0.1	UDP	65 58312 → 20002 Len=21
19	1.626741201	127.0.0.1	127.0.0.1	UDP	53 20002 → 58312 Len=9
20	1.626769291	127.0.0.1	127.0.0.1	UDP	66 52892 → 20001 Len=22
21	1.626868932	127.0.0.1	127.0.0.1	UDP	53 20001 → 52892 Len=9
22	1.626889505	127.0.0.1	127.0.0.1	UDP	53 20000 → 58227 Len=9

그림 15: 모든 UDP Packet

TCP Protocol과 비교할때 아주 작은양의 Packet이다.  
1단계 Client의 수식 전송

5	1.624851279	127.0.0.1	127.0.0.1	UDP	69 58227 → 20000 Len=25
Frame 5: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface any, id 0					
Linux cooked capture v1					
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1					
User Datagram Protocol, Src Port: 58227, Dst Port: 20000					
Data (25 bytes)					
Data: 28312b32292f322a372d35302b39362d2832372d3330292f35					
[Length: 25]					
0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....		
0010	45 00 00 35 24 1c 40 00	40 11 18 9a 7f 00 00 01	E..5\$.@. @.....		
0020	7f 00 00 01 e3 73 4e 20	00 21 fe 34 28 31 2b 32	.....sN .!..4(1+2		
0030	29 2f 32 2a 37 2d 35 30	2b 39 36 2d 28 32 37 2d	)/2*7-50 +96-(27-		
0040	33 30 29 2f 35		30)/5		

그림 16: ClientUdp와 InputParseUdpServer 수식 전송 Packet

UDP Datagram 정보는 별도의 연결과정없이 곧바로 전송한다.  
2단계 결과값 응답

22	1.626889505	127.0.0.1	127.0.0.1	UDP	53 20000 → 58227 Len=9
Frame 22: 53 bytes on wire (424 bits), 53 bytes captured (424 bits) on interface any, id 0					
Linux cooked capture v1					
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1					
User Datagram Protocol, Src Port: 20000, Dst Port: 58227					
Data (9 bytes)					
Data: 35372e303939393938					
[Length: 9]					
0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....		
0010	45 00 00 25 24 2d 40 00	40 11 18 99 7f 00 00 01	E..%\$.@. @.....		
0020	7f 00 00 01 4e 20 e3 73	00 11 fe 24 35 37 2e 30	.....N.s ...\$.57.0		
0030	39 39 39 39 38		99998		

그림 17: InputParseUdpServer의 결과값 응답 Packet

응답값 전송후 별도의 종료 과정 또한 없다.

### 3.2.2 InputParseUdpServer와 AddSubUdpServer / MultDivUdpServer

1단계 정의된 프로토콜로 수식 전송

8	1.625401100	127.0.0.1	127.0.0.1	UDP	64 38769 → 20002 Len=20
Frame 8: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface any, id 0					
Linux cooked capture v1					
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1					
User Datagram Protocol, Src Port: 38769, Dst Port: 20002					
Data (20 bytes)					
Data: 332e3030303030303020343720322e30303030303030					
[Length: 20]					
0000	00 00 03 04 00 06 00 00	00 00 00 00 00 00 08 00	.....		
0010	45 00 00 30 24 1f 40 00	40 11 18 9c 7f 00 00 01	E..0\$.@. @.....		
0020	7f 00 00 01 97 71 4e 22	00 1c fe 2f 33 2e 30 30	.....qN" .../3.00		
0030	30 30 30 30 20 34 37 20	32 2e 30 30 30 30 30 30	0000 47 2.000000		

그림 18: UDP 정의된 프로토콜로 수식 전송 Packet

2단계 계산값 응답

9	1.625560714	127.0.0.1	127.0.0.1	UDP	52	20002 → 38769	Len=8
▶ Frame 9: 52 bytes on wire (416 bits), 52 bytes captured (416 bits) on interface any, id 0 ▶ Linux cooked capture v1 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1 ▶ User Datagram Protocol, Src Port: 20002, Dst Port: 38769 ▼ Data (8 bytes)							
Data: 312e353030303030							
[Length: 8]							
0000	00	00	03	04	00	06	00 00 00 00 00 00 08 00
0010	45	00	00	24	24	20	40 00 40 11 18 a7 7f 00 00 01
0020	7f	00	00	01	4e	22	97 71 00 10 fe 23 31 2e 35 30
0030	30	30	30	30			0000

4. 소스 코드 및 참고 자료

4.1 소스 코드

<https://github.com/gbdngb12/NetworkTool/tree/main/NetworkHW%232>

4.2 참고 자료

- [https://blogofscience.com/Socket\\_Programming-KLDP.html](https://blogofscience.com/Socket_Programming-KLDP.html)
- <https://woongsios.tistory.com/288>
- <https://dev-with-precious-dreams.tistory.com/entry/%EC%9E%90%EB%A3%8C%E>
- [https://seongkyun.github.io/data\\_structure/2019/08/04/data\\_structure/](https://seongkyun.github.io/data_structure/2019/08/04/data_structure/)