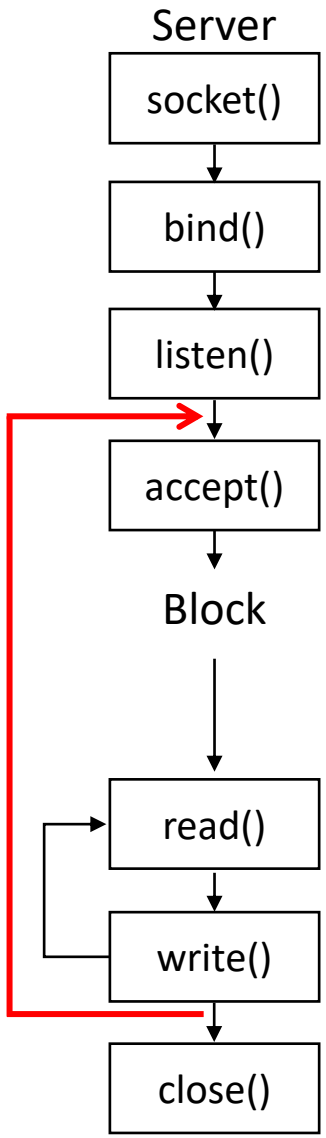
ISL

Network Programming #2

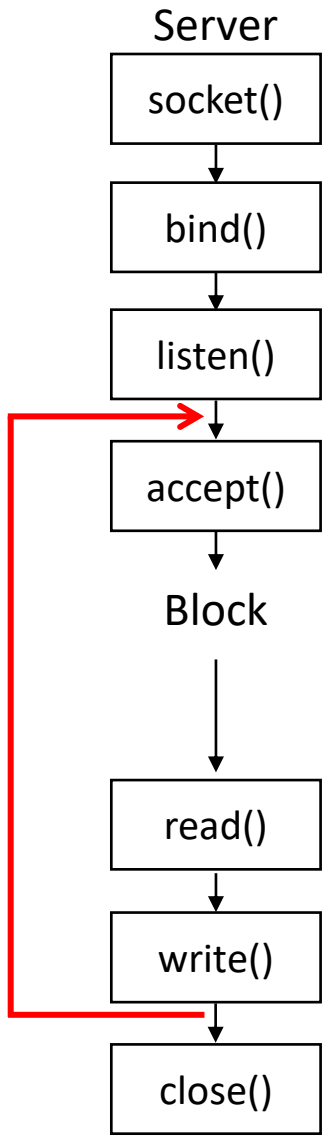
ISL (IoT Standard Lab)

Index

Iterative Server 흐름



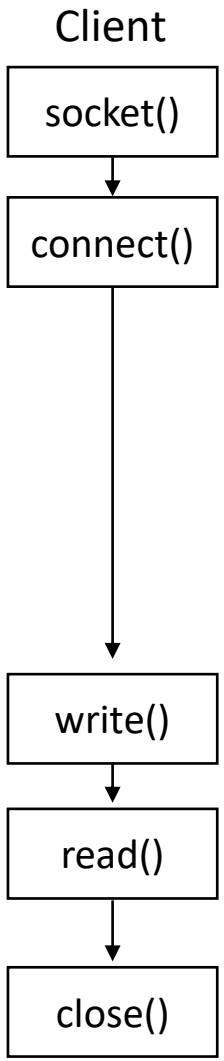
서버




```c
for(int i=0; i<5; i++) {
    if((cSockfd = accept(sockfd, (struct sockaddr *)&cliaddr, &len)) < 0) {
        perror("accept error");
        return -1;
    }

    int str_len;
    memset(buf, 0, sizeof(buf));
    while((str_len = read(cSockfd, buf, BUF_SIZE)) != 0)
        write(cSockfd, buf, str_len);

    printf("close client(%d)\n", i);
    close(cSockfd);
}

close(sockfd);
return 0;
```

1. read()는 디폴트 설정으로 읽을 데이터가 있을 때까지 Block
2. read()의 반환 값은 읽었던 문자의 길이
3. 상대방 socket이 close()하면 EOF=0을 반환

클라이언트



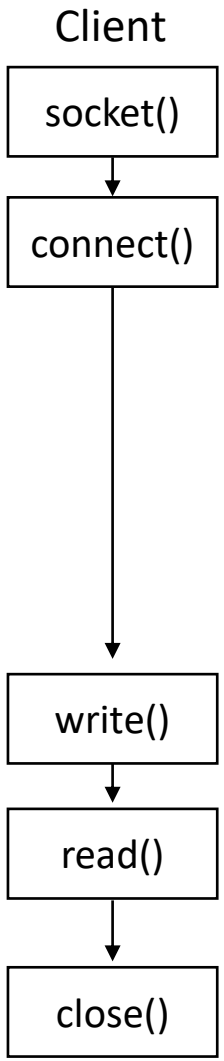

```c
if (connect(sockfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    return -1;
}

while(1) {
    fputs("Input message(Q to quit): ", stdout);
    fgets(buf, BUF_SIZE, stdin);

    if(!strcmp(buf, "q\n") || !strcmp(buf, "Q\n"))
        break;

    int str_len;
    write(sockfd, buf, strlen(buf));
    str_len = read(sockfd, buf, sizeof(buf));
    buf[str_len-1] = 0;

    printf("Message from server: %s\n", buf);
}

close(sockfd);
return 0;
```

fputs()는 파일에 문자열을 쓰는 함수
→ stdout(표준출력)으로 문자열 출력

fgets()는 파일에서 문자열을 읽는 함수
→ stdin(표준입력)에서 문자열 입력

01 Iterative Server

클라이언트




```c
if (connect(sockfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    return -1;
}

while(1) {
    fputs("Input message(Q to quit): ", stdout);
    fgets(buf, BUF_SIZE, stdin);

    if(!strcmp(buf, "q\n") || !strcmp(buf, "Q\n"))
        break;

    int str_len;
    write(sockfd, buf, strlen(buf));
    str_len = read(sockfd, buf, sizeof(buf));
    buf[str_len-1] = 0;

    printf("Message from server: %s\n", buf);
}

close(sockfd);
return 0;
```

stdin을 통해 받은 '\n' (엔터키) 제거

01 Iterative Server

실행 결과

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2$ ./server 8080
close client(0)
close client(1)
close client(2)
close client(3)
close client(4)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2$ ./client 127.0.0.1 8080
Input message(Q to quit): Hello
Message from server: Hello
Input message(Q to quit): How are you
Message from server: How are you
Input message(Q to quit): Bye
Message from server: Bye
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2$ ./client 127.0.0.1 8080
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2$ ./client 127.0.0.1 8080
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2$ ./client 127.0.0.1 8080
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2$ ./client 127.0.0.1 8080
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2$
```

프로그램 개요

계산기 프로그램 구현하기(어플리케이션 프로토콜)

서버는 클라이언트로부터 여러 개의 숫자와 연산자 정보를 전달받는다. 그러면 서버는 전달받은 숫자를 바탕으로 덧셈, 뺄셈 또는 곱셈을 계산해서 그 결과를 클라이언트에게 전달한다. 예를 들어서 서버로 3, 5, 9가 전달되고 덧셈연산이 요청된다면 클라이언트에는 3+5+9의 연산결과가 전달되어야 하고, 곱셈연산이 요청된다면 클라이언트에는 3×5×9의 연산결과가 전달되어야 한다. 단, 서버로 4, 3, 2가 전달되고 뺄셈연산이 요청되면 클라이언트에는 4-3-2의 연산결과가 전달되어야 한다. 즉, 뺄셈의 경우에는 첫 번째 정수를 대상으로 뺄셈이 진행되어야 한다.

이와 같은 서버 클라이언트 사이에서의 데이터 송수신 명세가 바로 프로토콜이다!

```
root@my_linux
root@my_linux:/tcpip# gcc op_client.c -o opclient
root@my_linux:/tcpip# ./opclient 127.0.0.1 9190
Connected...........
Operand count: 3
Operand 1: 12
Operand 2: 24
Operand 3: 36
Operator: +
Operation result: 72
```

위의 명세를 기반으로 하는 클라이언트 프로그램의 실행의 예

서버, 클라이언트의 구현

- 클라이언트는 서버에 접속하자마자 피연산자의 개수정보를 1바이트 정수형태로 전달한다.
- 클라이언트가 서버에 전달하는 정수 하나는 4바이트로 표현한다.
- 정수를 전달한 다음에는 연산의 종류를 전달한다. 연산정보는 1바이트로 전달한다.
- 문자 +, −, * 중 하나를 선택해서 전달한다.
- 서버는 연산결과를 4바이트 정수의 형태로 클라이언트에게 전달한다.
- 연산결과를 얻은 클라이언트는 서버와의 연결을 종료한다.

▶ 그림 05-1 : 클라이언트 op_client.c의 데이터 전송 포맷

프로토콜은 위와 같이(그 이상으로) 명확히 정의해야 한다.

op_server.c op_client.c를 통해 구현하였으니, 참조!

클라이언트


```c
if (connect(sockfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    return -1;
}
puts("Connected......");

printf("Operand count: ");
scanf("%d", &opCount);
buf[0] = (char)opCount;

for(int i=0; i<opCount; i++) {
    printf("Operand %d: ", i);
    scanf("%d", (int*)&buf[(i*4)+1]);
}

printf("Operator: ");
scanf(" %c", &buf[(opCount*4)+1]);
write(sockfd, buf, (opCount*4)+2);
//write(sockfd, buf, 1+(opCount*4)+1);

read(sockfd, &opResult, 4);
printf("Operation result: %d\n", opResult);

close(sockfd);
return 0;
```

1. opCount가 4바이트 정수이기 때문에 1바이트인 char로 타입캐스팅
2. 따라서 char 기준으로 128부터 음수가 됨 (overflow)
3. buf가 unsigned char라면 음수 없이 255까지 표현가능

02 Calculator program

클라이언트

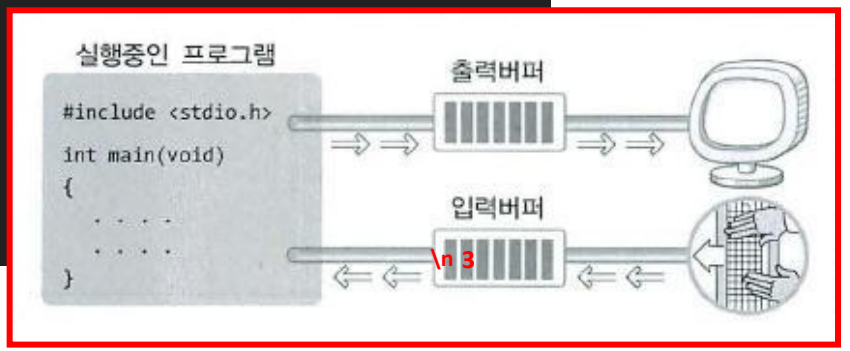
```c
#include <stdio.h>
int main() {
    int a;
    char b;
    unsigned char c;

    scanf("%d", &a);
    b = (char)a;
    c = (char)a;

    printf("signed   b = %d\n", b);
    printf("unsigned c = %d\n", c);
    return 0;
}
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./test
127
signed   b = 127
unsigned c = 127
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./test
128
signed   b = -128
unsigned c = 128
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./test
255
signed   b = -1
unsigned c = 255
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./test
256
signed   b = 0
unsigned c = 0
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./test
257
signed   b = 1
unsigned c = 1
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$
```

클라이언트


```c
if (connect(sockfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    return -1;
}
puts("Connected......");

printf("Operand count: ");
scanf("%d", &opCount);
buf[0] = (char)opCount;

for(int i=0; i<opCount; i++) {
    printf("Operand %d: ", i);
    scanf("%d", (int*)&buf[(i*4)+1]);
}

printf("Operator: ");
scanf(" %c", &buf[(opCount*4)+1]);
write(sockfd, buf, (opCount*4)+2);
//write(sockfd, buf, 1+(opCount*4)+1);

read(sockfd, &opResult, 4);
printf("Operation result: %d\n", opResult);

close(sockfd);
return 0;
```

1. scanf()에서 %c로 받으면 stdin에서 문자를 받을 수 있음
2. %c는 앞에서 scanf()를 여러 번 사용하고 남은 \n(엔터)가 남아 바로 출력되는 상황이 생김
   *(%d일 경우는 남아있는 \n 무시)
3. space + %c (%c)를 넣으면 \n를 무시하고 입력을 다시 받을 수 있음



02 Calculator program

클라이언트


```c
if (connect(sockfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    return -1;
}
puts("Connected......");

printf("Operand count: ");
scanf("%d", &opCount);
buf[0] = (char)opCount;

for(int i=0; i<opCount; i++) {
    printf("Operand %d: ", i);
    scanf("%d", (int*)&buf[(i*4)+1]);
}
```

```c
char buf[BUF_SIZE];
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ gcc client.c -o client
client.c: In function ‘main’:
client.c:43:17: warning: format ‘%d’ expects argument of type ‘int *’, but argument 2 has type ‘char *’ [-Wformat=]
   43 |         scanf("%d", &buf[(i*4)+1]);
      |                 ~^   ~~~~~~~~~~~~
      |                  |   |
      |                  |   char *
      |                 int *
```

02 Calculator program

클라이언트


```c
if (connect(sockfd, (const struct sockaddr*)&servaddr, sizeof(servaddr)) < 0) {
    perror("connect error");
    return -1;
}
puts("Connected......");

printf("Operand count: ");
scanf("%d", &opCount);
buf[0] = (char)opCount;

for(int i=0; i<opCount; i++) {
    printf("Operand %d: ", i);
    scanf("%d", (int*)&buf[(i*4)+1]);
}

printf("Operator: ");
scanf(" %c", &buf[(opCount*4)+1]);
write(sockfd, buf, (opCount*4)+2);
//write(sockfd, buf, 1+(opCount*4)+1);

read(sockfd, &opResult, 4);
printf("Operation result: %d\n", opResult);

close(sockfd);
return 0;
```

02 Calculator program

서버

```c
if((cSockfd = accept(sockfd, (struct sockaddr *)&cliaddr, &len)) < 0) {
    perror("accept error");
    return -1;
}

read(cSockfd, &opCount, 1);
printf("Operand count: %d\n", opCount);

for(int i=0; i<opCount; i++) {
    read(cSockfd, &operand[i], 4);
    printf("Operand %d: %d\n", i, operand[i]);
}

read(cSockfd, &operator, 1);
printf("Operator: %c\n", operator);

int result = 0;
switch (operator) {
case '+':
    for(int i=0; i<opCount; i++) {
        result += operand[i];
    }
    break;
case '-':
    for(int i=0; i<opCount; i++) {
        result -= operand[i];
    }
    break;
case '*':
    for(int i=0; i<opCount; i++) {
        result *= operand[i];
    }
    break;
}

printf("Operation result: %d\n", result);
write(cSockfd, &result, 4);
```

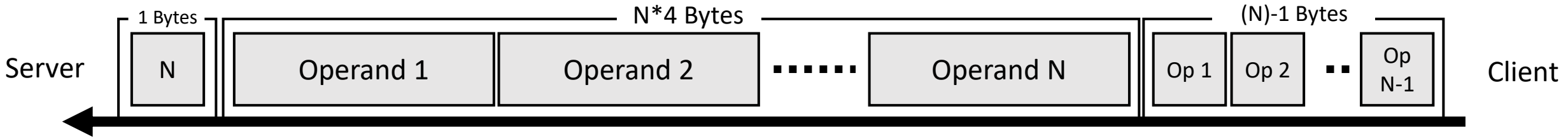
02 Calculator program

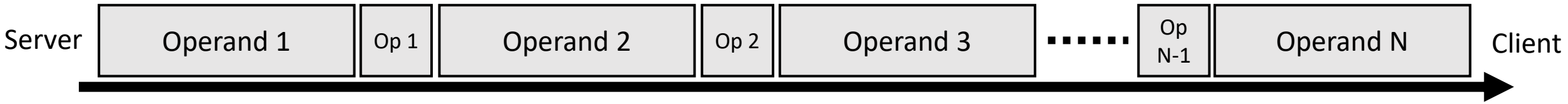
실행 결과

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./server 8080
Operand count: 3
Operand 0: 50
Operand 1: 25
Operand 2: 35
Operator: +
Operation result: 110
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$
```

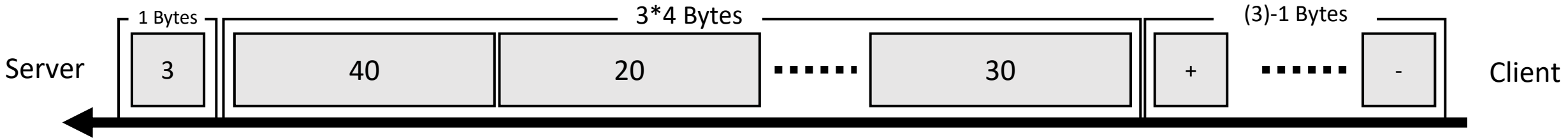
```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 127.0.0.1 8080
Connected......
Operand count: 3
Operand 0: 50
Operand 1: 25
Operand 2: 35
Operator: +
Operation result: 110
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$
```

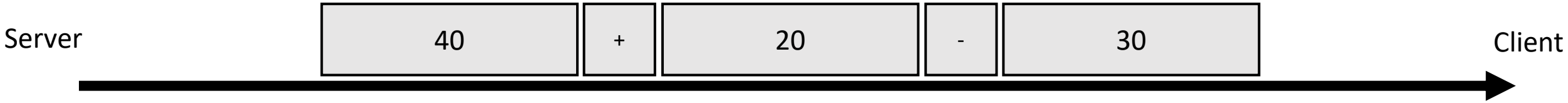
Advanced Calculator Program



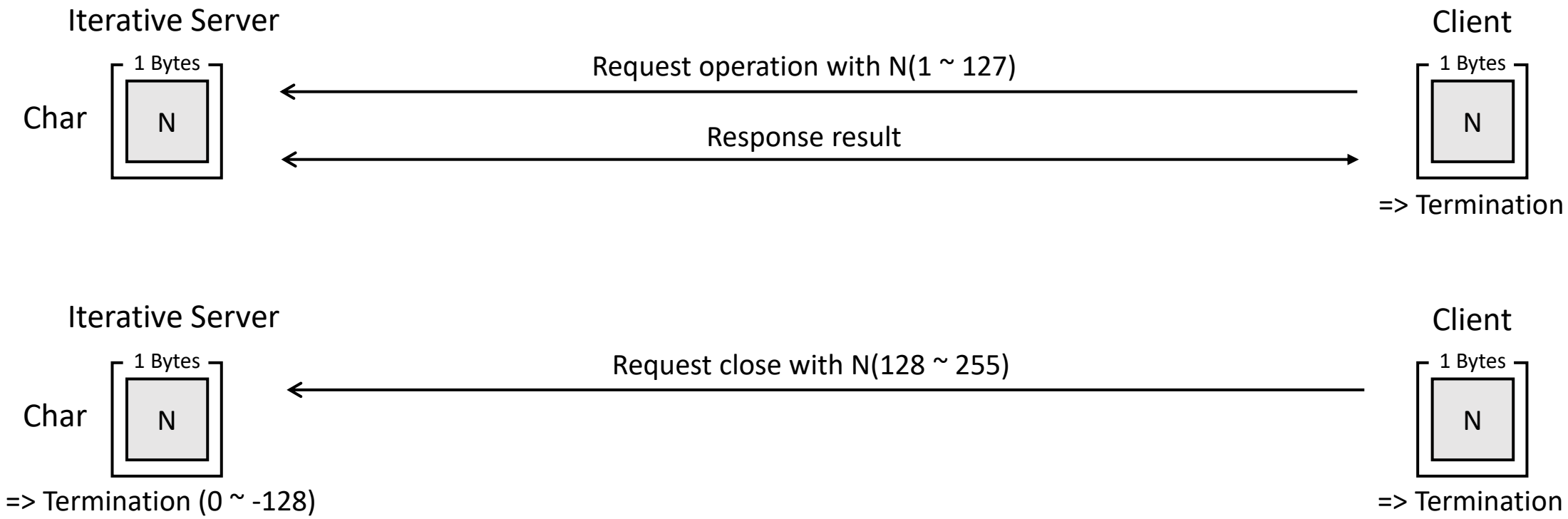


Advanced Calculator Program





Advanced Calculator Program



Advanced Calculator Program

- **client.c**

1. 클라이언트 실행 시 main 함수의 매개변수로 포트번호와 서버의 IP주소를 받아서 실행

   Ex) ./client 8080 127.0.0.1 (순서 확인) – 지난번 과제와 같음

2. 매개변수를 활용하여 서버에게 연결 요청
3. 표준입력을 통해 operand count와 이 수만큼 operand를 받고 (operand count)-1 의 수만큼 operator(1바이트)를 입력 받음
   1. 이 때 operator는 +, -, *로 한정
   2. Operand count를 입력 받을 시 표준출력으로 *Operand count:* 를 출력
   3. Operand와 Operator 입력을 받을 시 표준출력으로 각 이름들과 함께 입력을 받는 순으로 0부터 번호를 같이 출력
      1. Operand 0: ,Operand 1:
      2. Operator 0: ,Operator 1:
4. 표준입력으로 받은 데이터를 char 배열로 받아 한번에 전송
   1. 서버에게 operand count를 1바이트로 보내고 이 수만큼 operand를 4바이트로 전송하고, (operation count)-1 의 수만큼 operator(1바이트)를 전송

   Ex) (3 | 4, 5, 7 | +, -) 형태를 char 배열을 통해 한번에 write()로 전송
5. 결과를 받게 되면 이를 표준출력으로 Operation result: 와 함께 출력하고 소켓을 닫고 종료
6. operand count를 전송할 때 char 기준 0보다 작거나 같은 값을 전송하는 케이스의 경우, 이 값을 서버에게 전송하고 추가적인 표준입력없이 소켓을 닫고 종료

Advanced Calculator Program

- **server.c**

1. 서버 실행 시 main 함수의 매개변수로 포트번호를 받아서 실행

   Ex) ./server 8080

2. INADDR_ANY와 매개변수로 받은 포트번호로 소켓 바인드 진행
3. 서버는 Iterative Server 형태로 구현하고 종료 조건은 다음과 같음
   1. 클라이언트가 1바이트로 전송한 operand count 정보를 char 형태로 저장
   2. operand count 정보가 char 기준으로 0보다 작거나 같은 숫자가 나온다면 서버는 반복을 빠져나와서 소켓을 닫고 프로그램 종료
      1. 이 때 server close(operand count)형태로 표준 출력을 하고 종료
4. 서버는 클라이언트가 전송한 데이터를 기반으로 데이터를 아래와 같이 연산함
   1. 클라이언트의 operand count 수만큼 operand를 받고 (operand count)-1 수만큼 연산자를 받음
   2. 클라이언트 전송한 operand를 앞에서부터 차례대로 operator로 계산하며, 이 때 *의 연산순위를 우선적으로 두지 않고 앞에서부터 계산

      Ex) [3 | 4, 5, 7 | +, -] > [4 + 5 – 7], [3 | 4, 5, 7 | +, *] > [(4 + 5) * 7]
5. 계산된 결과를 클라이언트에게 전송
   1. 전송하기 전에 클라이언트에서 출력된 결과와 똑같이 표준 출력을 하고 전송
   2. 프로그램 종료 조건에서는 똑같을 필요 없음
6. 반복된 클라이언트 요청에도 문제없이 결과가 전송되어야 함

Advanced Calculator Program 실행 결과

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./server 8080
Operand count: 3
Operand 0: 40
Operand 1: 50
Operand 2: 30
Operator 0: +
Operator 1: -
Operation result: 60
Operand count: 5
Operand 0: 60
Operand 1: 400
Operand 2: 2450
Operand 3: 4900
Operand 4: 100
Operator 0: +
Operator 1: +
Operator 2: +
Operator 3: *
Operation result: 781000
Operand count: 2
Operand 0: 20
Operand 1: 10
Operator 0: -
Operation result: 10
Server close(-128)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 3
Operand 0: 40
Operand 1: 50
Operand 2: 30
Operator 0: +
Operator 1: -
Operation result: 60
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 5
Operand 0: 60
Operand 1: 400
Operand 2: 2450
Operand 3: 4900
Operand 4: 100
Operator 0: +
Operator 1: +
Operator 2: +
Operator 3: *
Operation result: 781000
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 2
Operand 0: 20
Operand 1: 10
Operator 0: -
Operation result: 10
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 128
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./server 8080
Operand count: 3
Operand 0: 4
Operand 1: 5
Operand 2: 7
Operator 0: +
Operator 1: *
Operation result: 63
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 3
Operand 0: 4
Operand 1: 5
Operand 2: 7
Operator 0: +
Operator 1: *
Operation result: 63
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./server 8080
Server close(0)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./server 8080
Server close(-128)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./server 8080
Server close(-127)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./server 8080
Server close(-1)
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 0
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 128
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 129
smalldragon@SD-DESKTOP:~/Workspace/socket/socket2/calculator$ ./client 8080 127.0.0.1
Operand count: 255
```

03 Network Programming Assignment

Advanced Calculator Program

- 참고사항

1. 서버가 의도한 것 이외의 값을 받는 케이스를 예외 처리할 필요 없음
2. Operand count가 0~255까지만 정상 동작하면 됨
3. 총 계산 결과 값이 int 형을 넘어가서 overflow가 발생할 경우도 예외 처리할 필요 없음
4. 과제에서 의도한 대로 데이터를 주고받고 이를 출력하는 방식이 아닌, 겉으로 출력 결과만 똑같이 보인다면 점수 없음

- 제출관련

1. 서버 프로그램은 server.c, 클라이언트 프로그램은 client.c로 명명하여 과제 진행
2. 빌드 시(gcc) Warning이 발생해서는 안됨
3. 제출 시 두 파일을 “자신의 학번.tar” 파일로 제출

   Ex) 2020324067.tar

   ~/Workspace/socket1/(server.c, client.c)

```
smalldragon@DESKTOP-PMPPMHH:~/Workspace$ tar cvf 2020324067.tar -C socket1 server.c client.c
server.c
client.c
```

압축파일명 폴더명 파일명 파일명

4. 과제는 10점 만점
5. 제출 기한: 2023.03.31(금) PM 11:59
6. 지각 제출 허용: 2023.04.04(화) PM 11:59 / 하루 늦을 때 마다 2점 씩 감점

   지각제출 시 보낼 이메일: eunjia24@gmail.com
7. 기한 안에 아예 제출을 하지 않았을 시 점수 없음