

***Xilinx Zynq FPGA, TI DSP, MCU***  
***기반의 프로그래밍 및 회로 설계***  
***전문가 과정***

**<리눅스 네트워크 프로그래밍>**  
**2018.04.04 - 30 일차**

**강사 - 이상훈**  
**[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)**

**학생 - 안상재**  
**[sangjae2015@naver.com](mailto:sangjae2015@naver.com)**

## 1. hi-hello 프로그램

- 클라이언트에서 서버에게 “hi” 를 전송하면 서버에서 클라이언트로 “hello”를 전송하고, 클라이언트에서는 전송받은 “hello”를 터미널 창에 출력함.
- 서버에서는 새로운 클라이언트가 접속할 때마다 **forkf** 를 통해 프로세스를 만들고, 자식 프로세스와 클라이언트가 통신하게 함.

### <서버 프로세스>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

#define BUF_SIZE 32

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void read_childproc(int sig)
{
    pid_t pid;
    int status;
    pid = waitpid(-1, &status, WNOHANG); // 자식 프로세스 처리
    printf("Removed proc id : %d\n", pid);
}

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock;
    si serv_addr, clnt_addr;
    pid_t pid;
    struct sigaction act;
    socklen_t addr_size;
    int str_len=0, state;
    char buf[BUF_SIZE] = {0};
    char buf1[BUF_SIZE] = "hello";
    char buf2[BUF_SIZE] = "hi";

    if(argc != 2)
    {
        printf("use : %s <port>\n", argv[0]);
        exit(1);
    }

    act.sa_handler = read_childproc;
```

```

sigemptyset(&act.sa_mask);
act.sa_flags = 0;
state = sigaction(SIGCHLD, &act, 0);

serv_sock = socket(PF_INET, SOCK_STREAM, 0);

if(serv_sock == -1)
    err_handler("socket() error");

memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(atoi(argv[1]));

if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("bind() error");

if(listen(serv_sock, 5) == -1)
    err_handler("listen() error");

for(;;)
{
    addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (sap)&clnt_addr, &addr_size);

    if(clnt_sock == -1)
        continue;
    else
        puts("New Client Connected ...");

    pid = fork();
    if(pid == -1)
    {
        close(clnt_sock);
        continue;
    }
    if(pid == 0)
    {
        close(serv_sock);

        for(;;)
        {
            str_len = read(clnt_sock, buf, BUF_SIZE);
            if(!strcmp(buf, buf2) && str_len) // read한 데이터가 "hi"와 같다면
            {
                printf("%s\n", buf);
                write(clnt_sock, buf1, sizeof(buf1)); // "hello" 전송
            }
        }

        close(clnt_sock);
        puts("Client Disconnected ... ");
        return 0;
    }
}
else

```

```

        close(clnt_sock);
    }
    close(serv_sock);
    return 0;
}

/////////////////////////////////////////////////////////////////
<클라이언트 프로세스>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

#define BUF_SIZE 32

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void read_routine(int sock, char *buf)
{
    int i;
    char buf1[BUF_SIZE] = "hello";

    while(1)
    {
        int str_len;

        str_len = read(sock, buf, BUF_SIZE); // 소켓파일로부터 데이터를 read함

        if(!strcmp(buf, buf1)) // read한 데이터가 "hello"와 같은지 체크
        {
            printf("\n\nmessage from server : %s", buf); // "hello"와 같으면 출력
        }
    }
}

void write_routine(int sock, char *buf)
{
    char buf1[BUF_SIZE] = "hi";
    char str_len;
    for(;;)
    {
        str_len = read(0, buf, BUF_SIZE); // 표준입력으로부터 입력을 받음

        if(!strcmp(buf, "q\n") || !strcmp(buf, "Q\n")) // 'q' 또는 'Q'를 입력받으면 종료함
        {

```

```

        shutdown(sock, SHUT_WR);
        return ;
    }
    if(!strncmp(buf, buf1, 2)) /* “hi”를 입력받았다면 소켓파일에 전송함(“hi”보다 많은 문자열을 입
력받으면, 다음번에 “hi”를 입력받아도 저번에 2번째 이상의 인덱스에 입력받은 문자 때문에 “hi”와 달라질 수 있다. 그
러므로 strcmp 함수를 사용해서 2개의 문자열만 비교함) */
        write(sock, buf1, strlen(buf1));
    }
}

int main(int argc, char **argv)
{
    pid_t pid;
    int i, sock;
    struct serv_addr;
    char buf[BUF_SIZE] = {0};

    if(argc != 3)
    {
        printf("use : %s <IP> <PORT>\n", argv[0]);
        exit(1);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0);

    if(sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() error");
    else
        puts("Connected .....");

    pid = fork(); // 프로세스를 2개를 만들어서 read와 write를 병렬로 수행함

    if(pid == 0)
        write_routine(sock, buf); // 자식은 write만 함
    else
        read_routine(sock, buf); // 부모는 read만 함

    close(sock);

    return 0;
}

```

### 1-1. 결과 분석

- 서버의 부모 프로세스는 `accept()` 함수에서 `blocking` 상태로 클라이언트의 접속을 기다리다가 새로운 클라이언트가 접속하면 `fork` 를 통해 자식 프로세스를 만든다. 클라이언트 1 개 당 서버의 자식 프로세스가 통신을 한다.

```
New Client Connected ...
hi
hi
hi
hi
hi
hi
hi
hi
New Client Connected ...
hi
hi
hi
hi
hi
hi
hi
hi
```

그림 1: 서버 프로세스

```
hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello

hi
message from server : hello
```

그림 2: 여러 개의 클라이언트 동시 접속

2. 클라이언트에서 입력하는 속도를 표현하는 프로그램.

- runtime : 다음 입력까지 걸리는 시간

cnt : 입력한 횟수

load\_ratio : 비율(횟수/시간)

#### <서버 프로세스>

```
#include "common.h"
#include "load_test.h"

#include <signal.h>
#include <sys/wait.h>

typedef struct sockaddr_in si;
typedef struct sockaddr * sp;

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void read_cproc(int sig)
{
    pid_t pid;
    int status;
    pid = waitpid(-1, &status, WNOHANG); // 종료된 자식 프로세스의 pid값을 반환받음
    printf("Removed proc id: %d\n", pid);
}

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock, len, state;
    char buf[BUF_SIZE] = {0};
    si serv_addr, clnt_addr;
    struct sigaction act;
    socklen_t addr_size;
    d struct_data;
    pid_t pid;

    if(argc != 2)
    {
        printf("use: %s <port>\n", argv[0]);
        exit(1);
    }

    act.sa_handler = read_cproc; // signal_handler 를 read_cproc()함수로 정의함
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    state = sigaction(SIGCHLD, &act, 0); // SIGCHLD를 받았을 때 signal_handler를 정의함

    serv_sock = socket(PF_INET, SOCK_STREAM, 0);

    if(serv_sock == -1)
        err_handler("socket() error");
```

```

memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(atoi(argv[1]));

if(bind(serv_sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("bind() error");

if(listen(serv_sock, 5) == -1)
    err_handler("listen() error");

for(;;)
{
    addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);

    if(clnt_sock == -1)
        continue;
    else
        puts("New Client Connected!\n");

    pid = fork();

    if(pid == -1)
    {
        close(clnt_sock);
        continue;
    }

    if(!pid)        // 자식 프로세스 수행
    {
        int cnt = 0;    // read한 횟수
        tv start, end;
        double runtime = 0.0;
        double load_ratio;

        close(serv_sock);

        for(;;)
        {
            gettimeofday(&start, NULL);    // 현재 시간 측정

            len = read(clnt_sock, (d *)&struct_data, BUF_SIZE); /* 소켓파일로부터 데이터
                                                                    이터를 read할 때까지 대기(blocking) */
            write(clnt_sock, (d *)&struct_data, len);

            gettimeofday(&end, NULL);    // read한 직후의 시간 측정

            runtime = get_runtime(start, end); // end- start의 시간을 측정(걸린 시간)
            cnt++;    // read한 횟수

            load_ratio = cnt / runtime;    // 비율을 구함

            printf("\nruntime = %f\n", runtime);
            printf("cnt = %d\n", cnt);
        }
    }
}

```



```

        printf("load_ratio = %lf\n", load_ratio);
    }

    close(clnt_sock);
    puts("Client Disconnected!\n");
    return 0;
}
else
    close(clnt_sock);
}
close(serv_sock);
return 0;
}

////////////////////////////////////
<클라이언트 프로세스>
#include "common.h"
#include <signal.h>
#include <setjmp.h>

jmp_buf env;
int tmp_sock;

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void read_proc(int sock, d *buf) // 소켓파일로부터 read한 데이터를 터미널에 출력함
{
    char msg[50];
    for(;;)
    {
        int len = read(sock, msg, BUF_SIZE);
        if(!len)
            return;

        msg[len] = 0;
        printf("msg from serv: %s\n", msg);
    }
}

void quit_proc(int signo) // SIGINT("CITL+C") 를 받으면 이동함
{
    printf("Exited!\n");
    shutdown(tmp_sock, SHUT_WR);
    longjmp(env, 1); // setjmp()가 위치한 행으로 이동
}

void write_proc(int sock, d *buf) // 키보드로부터 read한 데이터를 소켓 파일에 출력함
{
    char msg[32] = {0};

```

```

tmp_sock = sock;
signal(SIGINT, quit_proc);

for(;;)
{
    fgets(msg, BUF_SIZE, stdin);
    write(sock, msg, sizeof(msg));
}
}

int main(int argc, char **argv)
{
    pid_t pid;
    int i, sock;
    struct serv_addr;
    struct d_struct_data;
    char buf[BUF_SIZE] = {0};

    if(argc != 3)
    {
        printf("use: %s <IP> <port>\n", argv[0]);
        exit(1);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0);

    if(sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() error");
    else
        puts("Connected!\n");

    pid = fork();

    if(!pid)
    {
        int ret;

        if((ret = setjmp(env)) == 0)
            ;
        else if(ret > 0)
            goto end;

        write_proc(sock, (d *)&struct_data); // 자식 프로세스는 write만 함
    }
    else
        read_proc(sock, (d *)&struct_data); // 부모 프로세스는 read만 함
}

```

```

end:
    close(sock);
    return 0;
}

```

## 2-1. 결과 분석

- 클라이언트 프로세스의 입력속도를 빠르게 하면, 서버 프로세스의 load\_ratio 가 늘어남.

```

runtime = 0.164854
cnt = 34
load_ratio = 206.243100

runtime = 0.183030
cnt = 35
load_ratio = 191.225482

runtime = 0.178133
cnt = 36
load_ratio = 202.096187

runtime = 0.175949
cnt = 37
load_ratio = 210.288209

runtime = 0.192317
cnt = 38
load_ratio = 197.590437

runtime = 0.170199
cnt = 39
load_ratio = 229.143532

runtime = 0.173644
cnt = 40
load_ratio = 230.356361

runtime = 2.912937
cnt = 41
load_ratio = 14.075141

runtime = 1.231189
cnt = 42
load_ratio = 34.113365

```

그림 2: 서버 프로세스  
결과

3. 채팅 프로그램에서 채팅 도배를 식별하고, 블랙리스트를 관리하는 프로그램.

- 채팅 도배를 식별하는 방법은 서버 프로세스에서 클라이언트가 전송한 데이터를 read 하기 위해 대기하는 시간을 측정함. 그 시간이 일정 시간 이하이면 도배로 판별함.
- 도배를 10 번 이상 반복하면 블랙리스트에 등재되고 해당 클라이언트의 소켓파일은 종료됨.
- 한번 블랙리스트에 등재된 클라이언트의 IP 주소는 다음에 다시 접속이 불가능함.
- 클라이언트 서버에서 소켓파일에 데이터를 write 하는 방식은 수동/공격 모드가 있음.

### <서버 프로세스>

```

#include "load_test.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <stdbool.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_SIZE 128
#define MAX_CLNT 256

typedef struct sockaddr_in si;

```

```

typedef struct sockaddr *      sp;

int clnt_cnt = 0;
int clnt_socks[MAX_CLNT];
int cnt[MAX_CLNT];
pthread_mutex_t mtx;

// Black List
int black_cnt;
char black_list[MAX_CLNT][16];

// Information of Thread
typedef struct __iot{
    int sock;
    char ip[16];
    int cnt;
} iot;

iot info[BUF_SIZE];

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void proc_msg(char *msg, int len, int sock) // 브로드캐스트
{
    int i;

    pthread_mutex_lock(&mtx);

    for(i = 0; i < clnt_cnt; i++)
    {
        if(info[i].sock == sock) // 중복된 클라이언트는 write 안함
            continue;
        write(info[i].sock, msg, len);
    }

    pthread_mutex_unlock(&mtx);
}

void add_black_list(char *ip) // 블랙리스트에 클라이언트 ip 추가
{
    pthread_mutex_lock(&mtx);
    strcpy(black_list[black_cnt++], ip);
    printf("black_list = %s\n", black_list[black_cnt - 1]);
    pthread_mutex_unlock(&mtx);
}

bool check_black_list(char *ip)
{
    int i;

    pthread_mutex_lock(&mtx);

    printf("Here\n");

    for(i = 0; i < black_cnt; i++)

```

```

    {
        if(!strcmp(black_list[i], ip))
        {
            pthread_mutex_unlock(&mtx);
            return true;    // 기존 블랙리스트에 존재함
        }
    }

    pthread_mutex_unlock(&mtx);

    return false;
}

void *clnt_handler(void *arg)
{
    iot thread_info = *((iot *)arg);
    int len = 0, i;
    char msg[BUF_SIZE] = {0};

    tv start, end;
    double runtime = 0.0;
    double load_ratio;

    for(;;)
    {
        gettimeofday(&start, NULL);
        //len = read(clnt_sock, msg, sizeof(msg));
        len = read(thread_info.sock, msg, sizeof(msg));
        proc_msg(msg, len, thread_info.sock);    // 브로드캐스트
        gettimeofday(&end, NULL);

        runtime = get_runtime(start, end);    // 클라이언트의 입력 시간 측정
        load_ratio = 1.0 / runtime;
        printf("load_ratio = %f\n", load_ratio);

        if(load_ratio > 1.5)
            thread_info.cnt++;    // 도배 1번 추가

        if(thread_info.cnt > 10)    // 도배 10번 이상이면 블랙리스트 추가
        {
            write(thread_info.sock, "You're Fired!!!\n", 16);
            add_black_list(thread_info.ip);
            goto end;    // 블랙리스트가 되면 이동
        }
    }

end:
    pthread_mutex_lock(&mtx);

    /* 중복된 클라이언트 제거 */
    for(i = 0; i < clnt_cnt; i++)    // clnt_cnt는 현재 접속한 클라이언트 갯수
    {
        if(thread_info.sock == info[i].sock)
        {
            while(i++ < clnt_cnt - 1)
                info[i].sock = info[i + 1].sock;
            break;
        }
    }
}

```

```

    clnt_cnt--;    // 현재 접속한 클라이언트 수 1 감소
    pthread_mutex_unlock(&mtx);
    close(thread_info.sock);    // 블랙리스트 클라이언트는 종료

    return NULL;
}

int main(int argc, char **argv)
{
    int serv_sock, clnt_sock;
    struct serv_addr, clnt_addr;
    socklen_t addr_size;
    pthread_t t_id;
    int idx = 0;

    if(argc != 2)
    {
        printf("Usage: %s <port>\n", argv[0]);
        exit(1);
    }

    srand(time(NULL));

    pthread_mutex_init(&mtx, NULL);

    serv_sock = socket(PF_INET, SOCK_STREAM, 0);

    if(serv_sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("bind() error");

    if(listen(serv_sock, MAX_CLNT) == -1)
        err_handler("listen() error");

    for(;;)
    {
        addr_size = sizeof(clnt_addr);
        clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_addr, &addr_size);

        printf("Check Black List\n");

        if(check_black_list(inet_ntoa(clnt_addr.sin_addr))) // 블랙리스트에 등재되어있는지 체크
        {
            write(clnt_sock, "Get out of my server!!!\n", 23);
            close(clnt_sock);    // 블랙리스트에 등재되어있다면 해당 클라이언트 소켓파일은 종료
            continue;
        }

        pthread_mutex_lock(&mtx);

        info[clnt_cnt].sock = clnt_sock;
        strcpy(info[clnt_cnt].ip, inet_ntoa(clnt_addr.sin_addr));
    }
}

```

```

        info[clnt_cnt++].cnt = 0;

        pthread_mutex_unlock(&mtx);

        //pthread_create(&t_id, NULL, clnt_handler, (void *)&clnt_sock);
        pthread_create(&t_id, NULL, clnt_handler, (void *)&info[clnt_cnt - 1]);
        pthread_detach(t_id);
        /* 네트워크 바이트 순서의 주소 -> 인터넷 점 표준 표기법 */
        printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));

    }

    close(serv_sock);

    return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
<클라이언트 프로세스>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <sys/epoll.h>

#define BUF_SIZE      128
#define NAME_SIZE     32

typedef struct sockaddr_in  si;
typedef struct sockaddr *   sp;

char name[NAME_SIZE] = "[내가이긴다]";
char msg[2048];

void err_handler(char *msg)
{
    fputs(msg, stderr);
    fputc('\n', stderr);
    exit(1);
}

void make_rand_str(char *tmp)    // 공격모드로 설정했을 때, 소켓에 write할 데이터를 랜덤으로 정함 */
{
    int i, end = rand() % 7 + 3;

    for(i = 0; i < end; i++)
        tmp[i] = rand() % 26 + 65;
}

void *send_msg(void *arg)    // 서버에 메시지를 전송함
{
    int sock = *((int *)arg);
    char msg2[] = "https://kr.battle.net/heroes/ko/ <== 지금 당장 접속하세요!!\n";
    srand(time(NULL));

    char tmp1[32] = {0};

    for(;;)

```

```

    {
#ifdef PASSIVE /* 사용자가 수동으로 입력해서 소켓에 write함 */
        fgets(msg, BUF_SIZE, stdin);
        write(sock, msg, strlen(msg));
#endif

#ifdef ATTACK /* 사용자가 입력하지 않고 자동으로 데이터를 소켓에 write함 */
        make_rand_str(tmp1);

        printf("%s\n", msg);
        sprintf(msg, "%s %s %s", name, tmp1, msg2);
        printf("tmp1 = %s\n", tmp1);
        write(sock, msg, strlen(msg));
        sleep(5);
#endif
    }

    return NULL;
}

void *recv_msg(void *arg) // 서버로부터 입력받은 데이터를 터미널에 출력함
{
    int sock = *((int *)arg);
    char msg[NAME_SIZE + 2048];
    int str_len;

    for(;;)
    {
        str_len = read(sock, msg, NAME_SIZE + 2047);

        msg[str_len] = 0;
        fputs(msg, stdout);
    }

    return NULL;
}

int main(int argc, char **argv)
{
    int sock;
    struct serv_addr;
    pthread_t snd_thread, rcv_thread;
    void *thread_ret;

    sock = socket(PF_INET, SOCK_STREAM, 0);

    if(sock == -1)
        err_handler("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_addr.sin_port = htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() error");

    pthread_create(&snd_thread, NULL, send_msg, (void *)&sock);
    pthread_create(&rcv_thread, NULL, recv_msg, (void *)&sock);
    pthread_join(snd_thread, &thread_ret);
}

```



}

### 3-1. 결과 분석

- 클라이언트를 공격 모드로 설정하면 5 초에 한번 씩 임의의 데이터가 서버로 전송된다.
- 클라이언트를 수동 모드로 설정하고, 빠른 속도로 입력을 해서 도배를 11 번 이상 하면 블랙리스트에 추가된다.

```

Check Black List
Here
Connected Client IP: 127.0.0.1
load_ratio = 83333.333333
load_ratio = 0.199990
load_ratio = 0.199988
load_ratio = 0.199985
load_ratio = 0.199984
load_ratio = 0.199992
load_ratio = 0.199991
load_ratio = 0.199989
load_ratio = 0.199985
load_ratio = 0.199990
load_ratio = 0.199986
load_ratio = 0.199982
load_ratio = 0.199994
load_ratio = 0.199984
load_ratio = 1.186683
load_ratio = 200000.000000
load_ratio = 500000.000000
load_ratio = 500000.000000
load_ratio = 500000.000000
load_ratio = 500000.000000
load_ratio = 500000.000000
load_ratio = 333333.333333
load_ratio = 333333.333333
load_ratio = 333333.333333
load_ratio = 500000.000000
black_list = 127.0.0.1
^C

[내가이긴다] SC5IIAAT https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = TXSIIAAT
[내가이긴다] TXSIIAAT https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = SCSIIAAT
[내가이긴다] SCSIIAAT https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = RFFIIAAT
[내가이긴다] RFFIIAAT https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = IOMUEQHT
[내가이긴다] IOMUEQHT https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = FCJOMTMM
[내가이긴다] FCJOMTMM https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = SJOOMTMM
[내가이긴다] SJOOMTMM https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = CNHEFUMM
[내가이긴다] CNHEFUMM https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = MBEXFUMM
[내가이긴다] MBEXFUMM https://kr.battle.net/heroes/ko/ <=== 지금 당장 접속하세요 !!
tmp1 = SRVLK5CT
^C
sangjaeahn@sangjaeahn-900XSN:~/code/linux/network/4.4/prevent_dobae_complete$ gcc -o
PASSIVE -lpthread

```

그림 1: 클라이언트를 공격 모드로 설정했을 때 결과(좌: 서버, 우: 클라이언트)

```

sangjae@sangjae:~$ ./code/linux/network/4.4/prevent_dosae_completes.py ./serv 7777
Check Black List
Here
Connected Client IP: 127.0.0.1
load_ratio = 0.659159
load_ratio = 1.067326
load_ratio = 1.421498
load_ratio = 4.525747
load_ratio = 6.432068
load_ratio = 7.016805
load_ratio = 6.386675
load_ratio = 5.885296
load_ratio = 8.652016
load_ratio = 7.047812
load_ratio = 7.493499
load_ratio = 6.598178
load_ratio = 6.564221
load_ratio = 6.820073

```

그림 2: 클라이언트를 수동 모드로 설정했을 때 결과(좌: 서버, 우: 클라이언트)