시스템 프로그래밍 실습 Report

실습 제목: Proxy #2-1

Socket Programming

실습일자: 2022년 4월 11일 (월)

제출일자: 2022년 4월 27일 (수)

학 과: 컴퓨터정보공학부

담당교수: 최상호 교수님

실습분반: 목요일 7,8교시

학 번: 2018202065

성 명: 박 철 준

- Introduction

1. 제목

Proxy 2-1 Socket Programming

2. 목표

가. 다수의 client를 처리하는 server와 client를 구현하여야 한다.

나. Main server process는 client로부터의 통신 요청을 수락 후 client의 URL request 처리를 위한 새로운 프로세스("Sub process")를 생성하고 관리할 수 있어야 한다.

다. client로부터 통신 요청이 들어오면 Sub process 생성하여야 한다.

라. Sub server process는 client로부터 url request를 받아 Assignment 1-2의 연산을 수행하여야 한다.

마. Client connect와 disconnect를 알리는 정보를 아래와 같이 출력하여야 한다.

[client IP : client port number] client was connected

[client IP : client port number] client was disconnected

client IP는 127.0.0.1로 표기

바. Proxy 1-2의 연산을 수행하는데 Hashing, Check HIT or MISS(결과를 client에게 전송), Manipulate cache directory, Logging을 행하여야한다.

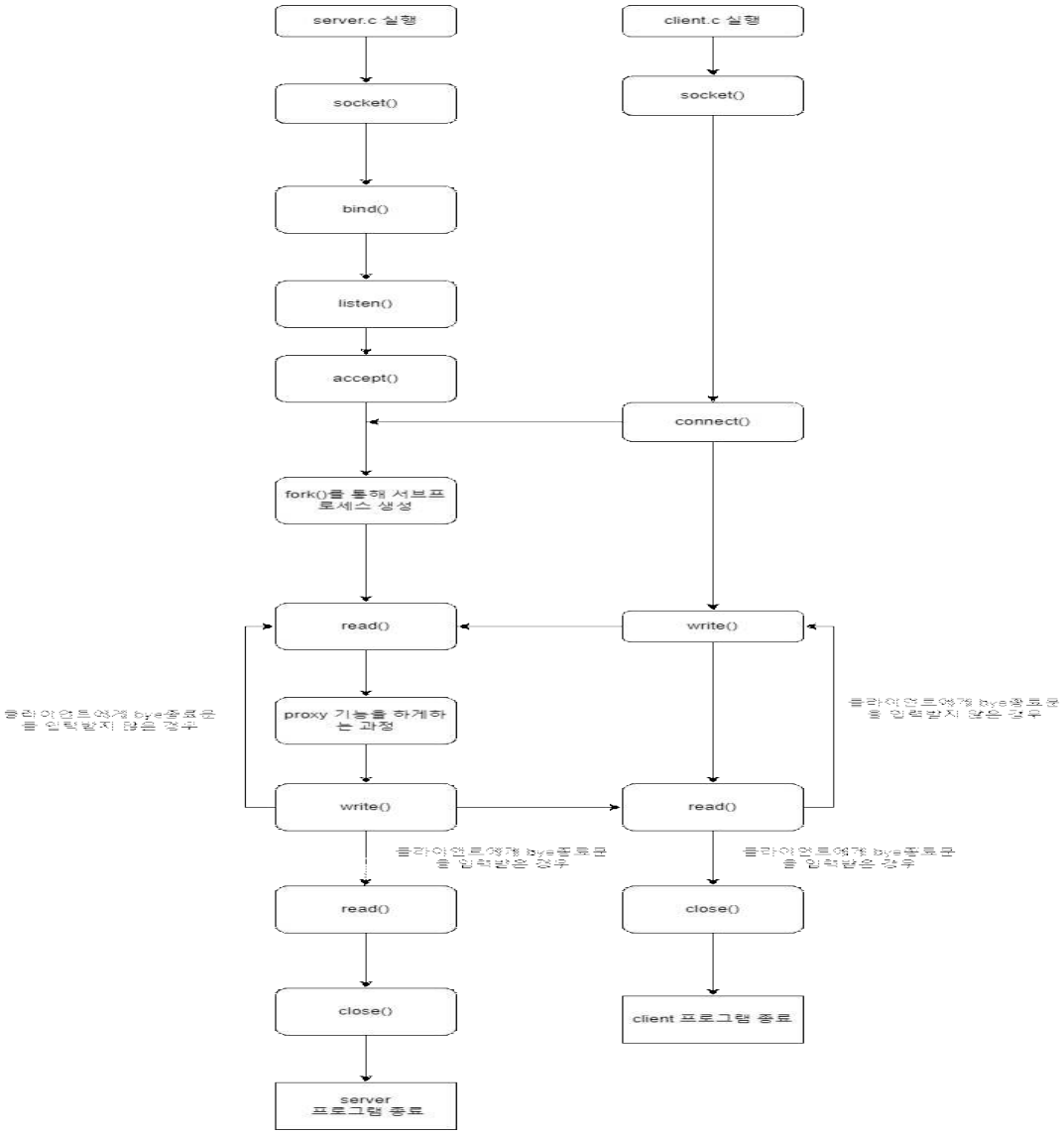
사. 실행 이후 client로부터의 URL 요청을 대기, client disconnect 시 프로세스 종료하여야 한다.

● 실습 결과

1 proxy #1-2

-Flow Chart

전체적인 프로그램의 흐름도는 아래와 같으며 proxy 1-2에서 작성한 main 함수는 이번 과제의 sub_server_processing_helper 함수가 되어 sub_process 함수로 이름을 변경하였고 새로운 main 함수를 설계하였다. sub_server_processing_helper 함수의 기능은 1-2과제에서 구현한 main 함수의 기능이다. 차이점이 있다면 위 함수를 사용하는 상위 함수 즉 main함수의 차일드 프로세스에서 클라이언트에게 bye라는 종료문이 들어오기 전까지 리퀘스트
를 반복해서 받기 때문에 기존에 1-2의 main 함수에서 구현한 것과 달리 위 함수에서 무한 반복문을 사용하지 않고 위 함수를 호출하는 main 함수의 차일드 프로세스에서
리퀘스트의 반복을 통해서 무한이 실행하는 차이점이 있다. 즉 sub_server_processing_helper 함수에서는 cache 적중 판단의 과정을 단 한 번 수행한다.



-Pseudo code
server.c의 main 함수

```
int main()
{
        //main 함수로 리눅스에서 실행 되며 지금까지 과제에서 정의한 함수들을
        //이용하여 본 과제의 목적인 socket programing을 구현한다.
        //proxy 서버통신에 있어 서버 역할을 한다.
        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 (socket함수 실행)
        {
                오류이면
                server : Can't open stream socket을 출력
                return 0;
        }

        if (bind함수 실행)
        {
                오류이면
                server : Can't bind local address을 출력
                close(socket_fd);
                return 0;
        }
        listen()함수 실행
        signal(SIGCHLD, (void*)handler);
        while (1)// bye종료문 request를 받을 때 까지 실행
        {
                bzero((char*)&client_addr, sizeof(client_addr));
                len = sizeof(client_addr);
                client_fd = accept함수를 통해클라이언트의 컨넥트를 대기한다.
                if (client_fd < 0 accept가 제대로 되었는지 확인)
                {
                        오류이면
                        Server : accept failed을 출력
                        close(socket_fd);
                        return 0;
                }
```

```
printf("[%s : %d] client was connected₩n", inet_ntoa(client_addr.sin_addr), client_addr.sin_port);
pid = fork()함수를 실행하여
클라이언트가 컨넥트 되면 차일드 프로세스생성 즉 서브 프로세스생성.
//다중 클라이언트를 위해 사용이 가능하다.

if (pid == -1)
{
        close(client_fd);
        close(socket_fd);
        continue;
}
if (pid == 0)
{
        //차일드 프로세스 즉 서브 프로세스의 실행이다.
        time_t start, end; //서브프로세스의 실행 시간을 기록하기 위해 선언
        time(&start); //서브프로세스의 시작시간을 저장
        int subprocess_misscount = 0;
        int subprocess_hitcount = 0;
        while ((len_out = read(client_fd, buf, BUFFSIZE)) > 0)// bye 종료문이 들어올 때 까지 클라이언트에게 리퀘스트를 받음
        {
                char* original_buf = buf;
                char * modified_buf = strtok(buf, "₩n");//개행문자를 없애준다.
                if (!strcmp(modified_buf, "bye") 종료문이 입력되면)
                {
                        break;
                }
                int check = sub_server_processing_helper(modified_buf) 함수를 통해 1-2에서 구현한 proxy기능을 수행한다.
                check변수에 리턴값을 저장함으로 hit과 miss를 판별
                if (check> 0 리퀘스트에 대해 hit인 경우)
                {
                        subprocess_hitcount++;
                        write(client_fd, "HIT₩n", sizeof("HIT₩n")); //클라이언트에게 hit을 전달함
                }
                else if (check == 0 리퀘스트에 대해 miss인 경우)
                {
```

```
                                        subprocess_misscount++;
                                        write(client_fd, "MISS₩n", sizeof("MISS₩n"));//클
                                        라이언트에게 miss를 전달함
                                }

                        }
                        //bye종료문이 입력되어 서브 프로세스를종료한다.
                        time(&end);//종료시간을 기록
                        int result = 0;
                        result = (int)(end - start);// 수행시간을 기록
                        bye(subprocess_hitcount, subprocess_misscount, result);// 서브 프
                        로세스의 종료문을 출력하기위한 함수 실행
                        printf("[%s : %d] client was disconnected₩n",
                        inet_ntoa(client_addr.sin_addr), client_addr.sin_port);
                        close(client_fd);
                        exit(0);
                }
                close(client_fd);

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

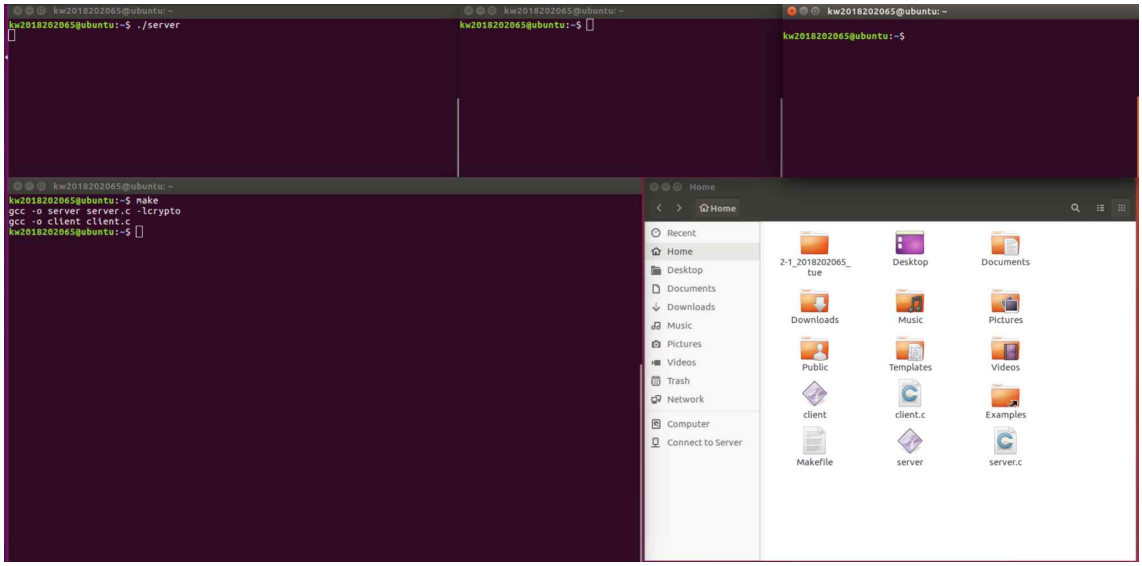
client.c의 main 함수

```
int main()
{
        // proxy 서버통신에 있어 클라이언트 역할을 한다.
        int socket_fd, len;
        struct sockaddr_in server_addr;
        char haddr[] = "127.0.0.1";
        char buf[BUFFSIZE];
        if ((socket()함수를 실행)
        {
                오류이면
                printf("can't create socket.₩n");
                return -1;
        }
        bzero(buf, sizeof(buf));
        bzero((char*)&server_addr, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr(haddr);
        server_addr.sin_port = htons(PORTNO);
        if (connect()함수를 통해 server와 컨넥트 시킴))
        {
                오류이면
                printf("can't connect.₩n");
                return -1;

        }
        write(STDOUT_FILENO, "input url > ", sizeof("input url > "));
        while (read(STDIN_FILENO, buf, sizeof(buf))를 통해 클라이언트에게 리퀘스트를 입
        력받음)
        {
                if (strcmp(buf, "bye₩n") == 0)// bye종료문을 받으면 소켓통신을 종료함
                {
                        write(socket_fd, buf, strlen(buf));// bye종료문을 받으면 소켓통신
                        을 종료함
                        break;

                }
                if (write(socket_fd, buf, strlen(buf)) > 0)//서버에게 리퀘스트를 전달.
                {
                        if ((len = read(socket_fd, buf, sizeof(buf))) > 0)//서버에게
                        response(hit or miss)를 받음
```

```
                {
                        write(STDOUT_FILENO, buf, strlen(buf));//받은 response를
                        출력
                        bzero(buf, sizeof(buf));//버퍼 초기화
                }
        }
        write(STDOUT_FILENO, "input url > ", sizeof("input url > "));
    }
    close(socket_fd);
    return 0;
}
```

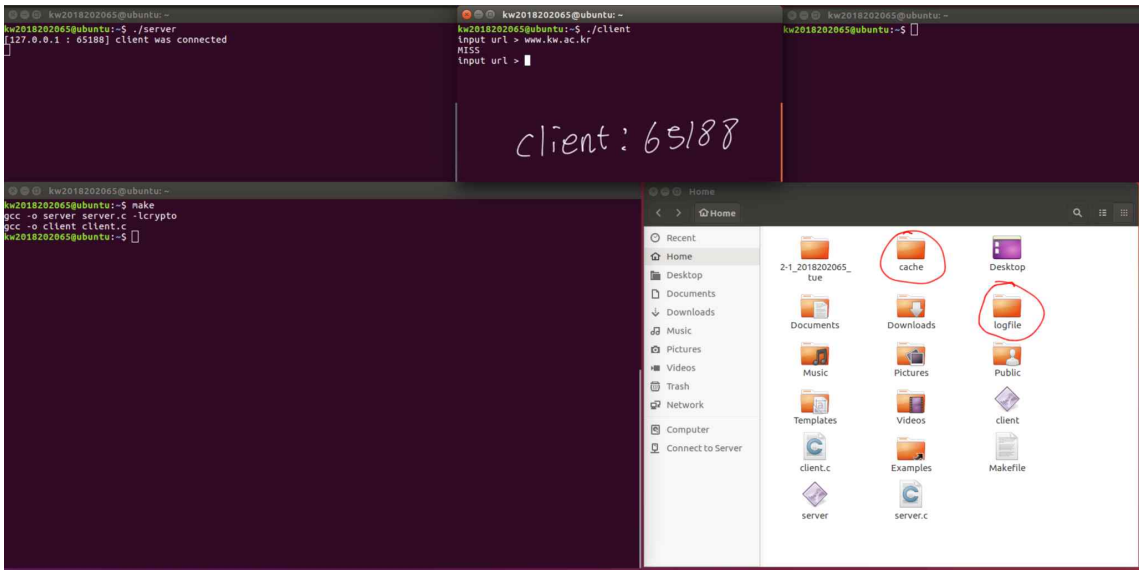
-결과 화면

1.



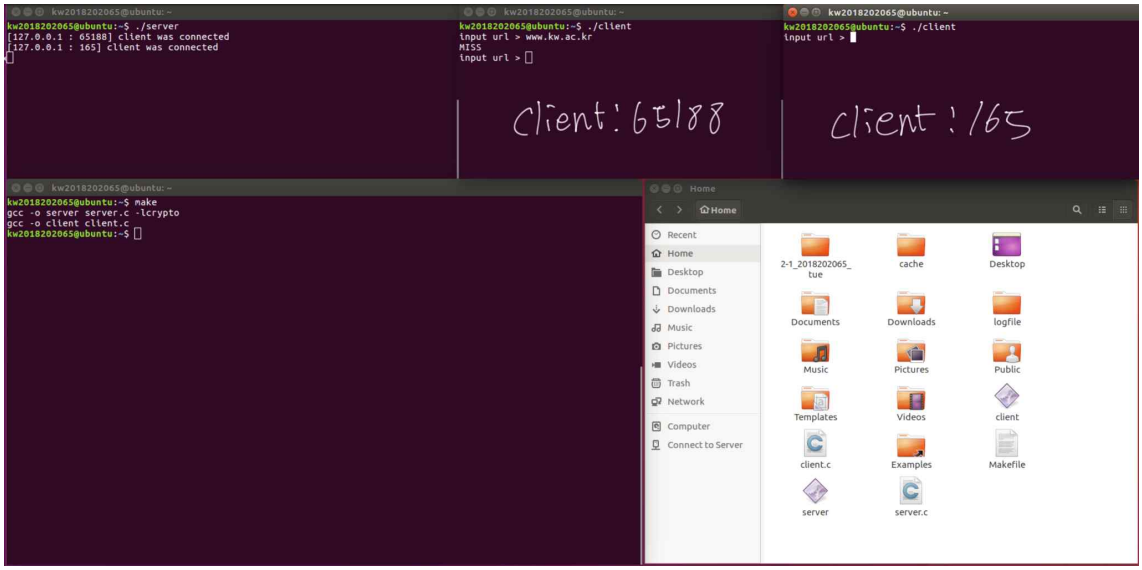
make파일로 컴파일 후 server를 켠 후의 모습

2.



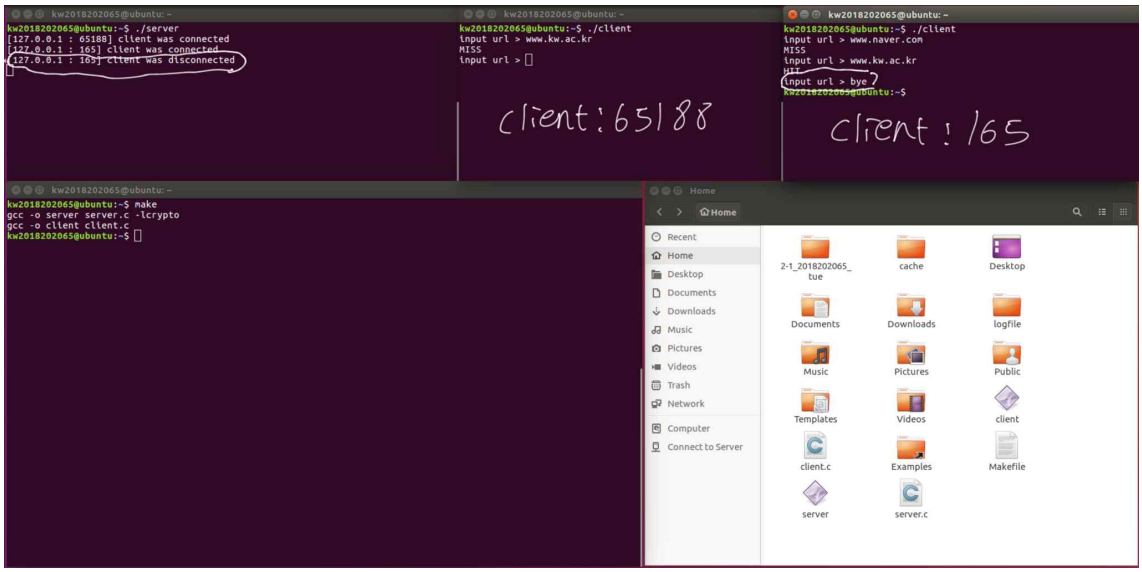
clinet:65188를 실행하여 www.kw.ac.kr을 입력하여 cache와 logfile 디렉토리를 생성한 모습이다. server터미널에는 65188이 connect되었음을 알려준다. 또한 입력받은 URL에 대해 miss response를 서버로부터 받음을 알 수 있다.

3.



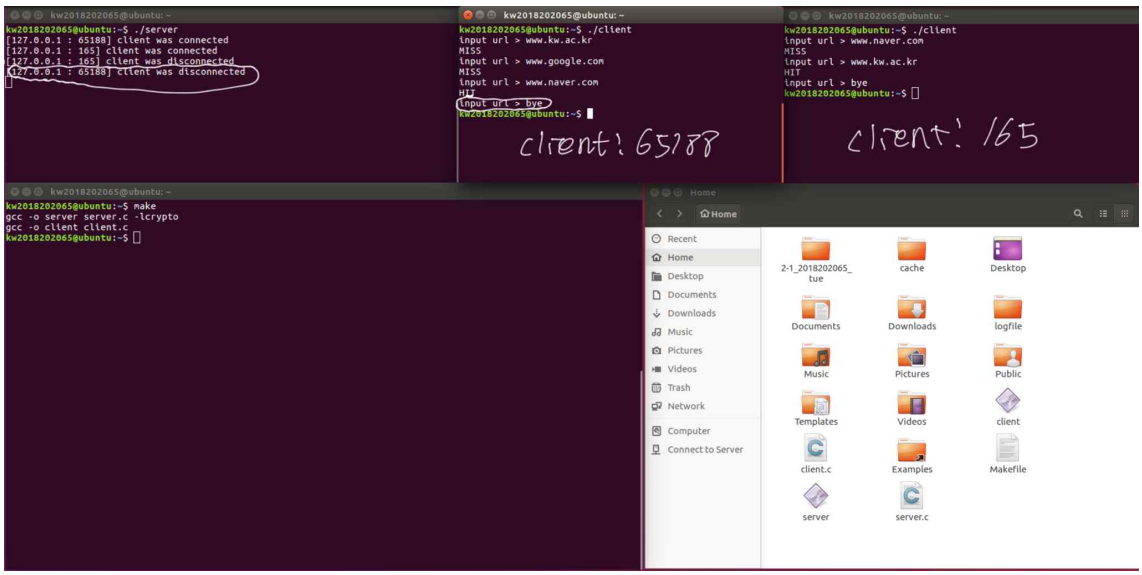
두 번째 client인 165를 실행한 모습니다. server터미널에는 165가 연결되었음을 알려준다.

4.



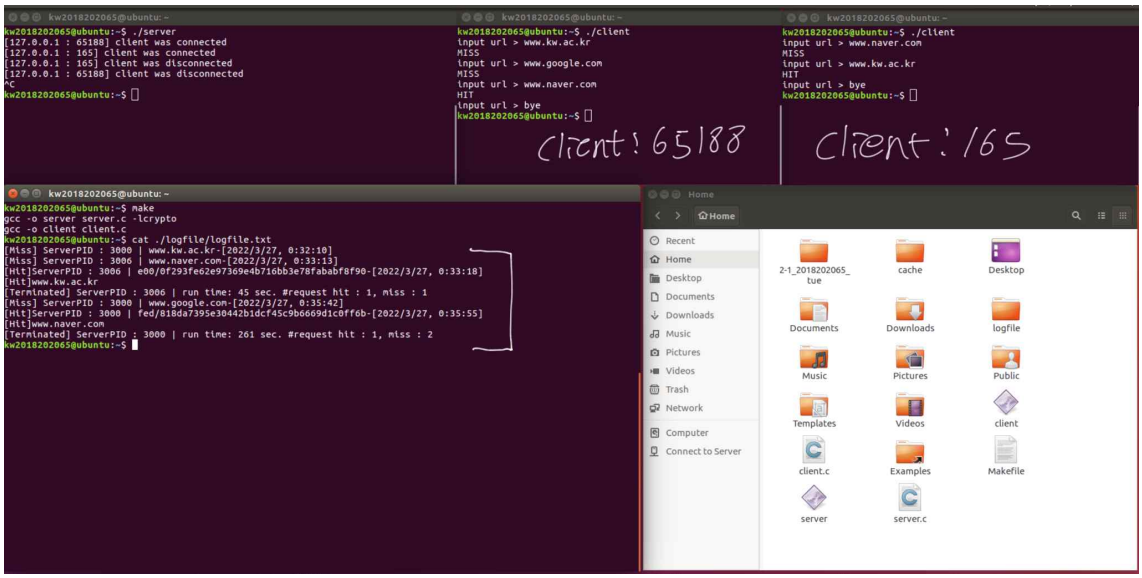
client 165는 www.naver.com을 입력했을 때 MISS를 www.kw.ac.kr을 입력했을 때 HIT response를 받음(client 65188로부터 입력을 받았기 때문)을 알 수 있다. 이후 bye를 통해 client를 종료시켰고 이에 대해 server의 터미널에도 알 수 있다.

5.



이후 아직 종료되지 않은 client 65188이 서버로 www.google.com과 www.naver.com의 request를 보냈고 이에 대해 서버로부터 MISS와 HIT(client 165로부터 입력을 받았기 때문)을 받음을 알 수 있다. 또한 bye종료문 request를 통해 클라이언트를 종료시켰다.

6.



^c를 통해 서버 프로세스를 종료하였고 logfile.txt를 확인한 결과 위의 사진의 시간 순서에 맞게 client:165가 소켓을 통해 연결된 서브 프로세스 아이디인 3006이 먼저 종료되고 이후 client:65188의 서브 프로세스가 종료됨을 알 수 있다. client 들이 하나의 cache를 공유하여 다중으로 실행되고 이에 따라 HIT, MISS가 잘 판별됨을 알 수 있다.

- 고찰

이번 과제를 진행하면서 과거 과제 1-3에서 proxy 서버 가장 큰 기능인 다중 사용자에 대한 캐시 처리에 대해 이해한 것을 이번 과제를 통해 실제 구현을 하며 멀티프로세싱과 소켓 프로그래밍을 더 심층적으로 이해할 수 있었다. 이번 과제를 진행하면서 가장 고민했던 부분은 기존의 1-2과제에서 구현한 cache 처리 기능은 1-2과제의 main 프로세스에서 무한 반복문을 사용하여 사용자에게 종료문을 입력받을 시 종료되는 형식으로 작성하였지만, 이번 과제에 있어서는 클라이언트가 서버에 접속되고 서버가 서브 프로세스를 생성하여 진행하기 때문에 어떤 부분에 무한 반복문을 해주어야 목표에 달성할 수 있을지 생각해 보았고 서버가 accept를 통해 클라이언트의 connect를 기다리고 클라이언트가 서버에 connect 되고 반복적으로 request를 보내기에 request를 보내는 부분을 반복으로 생각하고 코드를 구성하였고 1-2과제의 main 함수를 서브 프로세스에서 cache 과정을 도와주는 함수 (sub_server_processing_help)로 변경하고 무한 반복문을 없앰으로써 구현하였다.