시스템 프로그래밍 실습 Report

실습 제목: Proxy #2-2

Construction Proxy Connection

실습일자: 2022년 4월 18일 (월)

제출일자: 2022년 5월 04일 (수)

학 과: 컴퓨터정보공학부

담당교수: 최상호 교수님

실습분반: 목요일 7,8교시

학 번: 2018202065

성 명: 박 철 준

- Introduction

1. 제목

Proxy 2-2 Construction Proxy Connection

2. 목표

가. 이번 과제에서 Proxy server는 Web browser가 URL을 입력할 경우 HTTP request를 받아야한다.

나. Child process는 Web browser 의 요청에 응답 후 종료하여야 한다.

다. HTTP request header로부터 host 정보(URL) 추출하여야 한다.

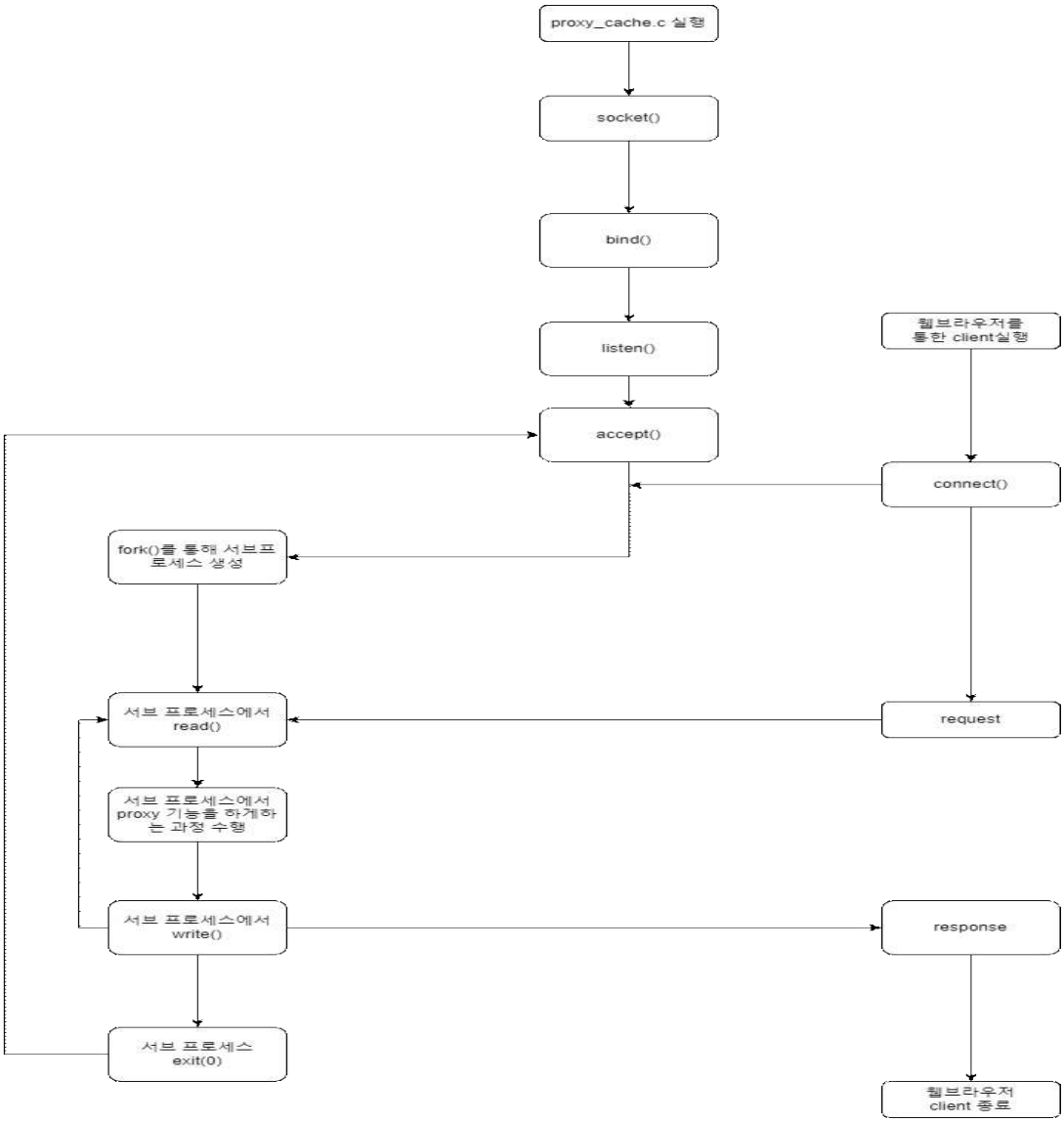
라. 추출한 host 정보를 Hashed URL로 변환 후 Cache Directory에서 HIT/MISS 판별하여야 한다.

● 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_process의 과정을 하도록 설계하였고 이에 따라 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 리퀘스트를 받으면 리스폰스 메시지와 함께 종료함으로 기존에 1-2의 main 함수에서 구현한 것과 달리 위 함수에서 무한 반복문을 사용하지 않고 위 함수를 호출하는 main 함수에서 차일드 프로세스에서
한번만 실행하고 종료되는 차이점이 있다. 즉 sub_server_processing_helper 함수에서는 cache 적중 판단의 과정을 단 한 번 수행한다.



-Pseudo code

proxy_cache.c의 main 함수

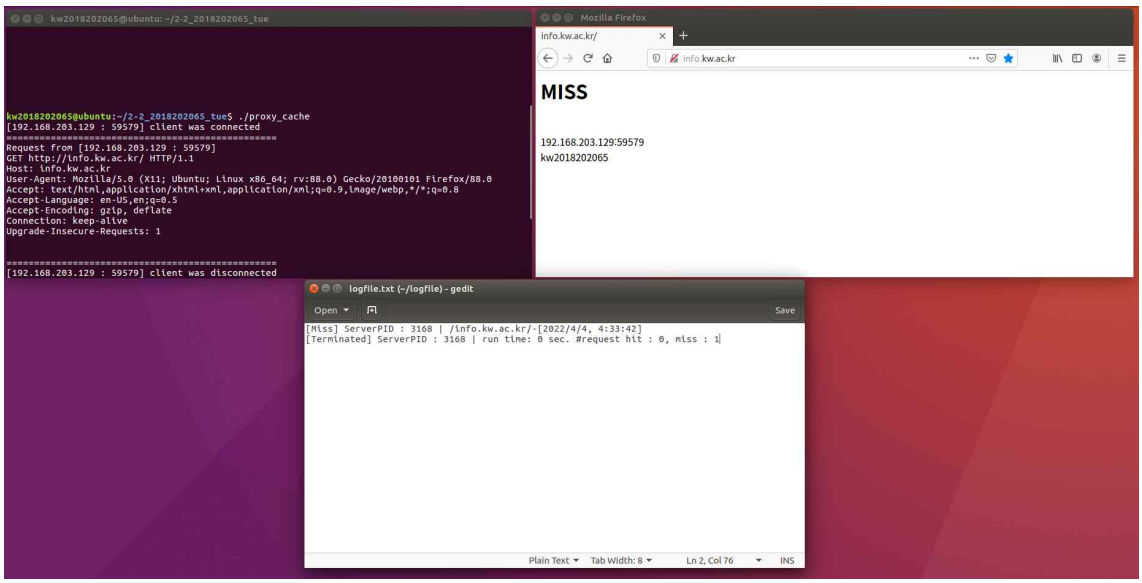
```
int main()
{
        //main 함수로 리눅스에서 실행 되며 위에 정의한 함수들을
        // 이용하여 본 과제의 목적인 Construction Proxy Connection을 구현한다.
        // proxy 서버통신에 있어 proxy 서버 역할을 한다.
        struct sockaddr_in server_addr, client_addr;
        int socket_fd, client_fd;
        int len, len_out;
        int status;
        pid_t pid;
        int opt = 1;
        if (socket함수 실행)
        {
                오류이면
                server : Can't open stream socket을 출력
                return 0;
        }
        setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));// bind 에
        러를 없애기 위해
        bzero((char*)&server_addr, sizeof(server_addr));
        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = htons(INADDR_ANY);
        server_addr.sin_port = htons(PORTNO);
        if (bind함수 실행)
        {
                오류이면
                server : Can't bind local address을 출력
                close(socket_fd);
                return 0;
        }
        listen(socket_fd, 5);//listen()함수를 실행하여 accept하기 전까지 block
        signal(SIGCHLD, (void*)handler);//좀비 프로세스를 예방하기 위해
        while (1)
        {
                struct in_addr inet_client_address;
                char buf[BUFFSIZE] = {0,};
                char response_header[BUFFSIZE] = { 0, };
                char response_message[BUFFSIZE] = { 0, };
                char tmp[BUFFSIZE] = { 0, };
```

```
char method[20] = { 0, };
char url[BUFFSIZE] = { 0, };
char * tok = NULL;
printf("pre accept ok!₩n");
len = sizeof(client_addr);
client_fd = accept(socket_fd, (struct sockaddr*)&client_addr,
&len);//accept함수를 통해클라이언트의 컨넥트를 대기한다.
if (lient_fd < 0 accept가 제대로 되었는지 확인)
{
        오류이면
        Server : accept failed을 출력
        return 0;

}
pid = fork();
if (pid == -1 fork 오류이면)
{
        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
        inet_client_address.s_addr = client_addr.sin_addr.s_addr;
        printf("[%s : %d] client was connected₩n",
        inet_ntoa(inet_client_address), client_addr.sin_port);
        read(client_fd, buf, BUFFSIZE);
        strcpy(tmp, buf);
        puts("========================================");
        printf("Request from [%s : %d]₩n", inet_ntoa(inet_client_address),
        client_addr.sin_port);
        puts(buf);//buf 출력
        puts("========================================");
        tok = strtok(tmp, " ");
        strcpy(method,tok)//tok한 문자가 Get 문자인지 판별을 위해 저장
        if (strcmp(method, "GET") == 0)// Get 메소드의 request일 때
        URL을 통해 cache생성
```

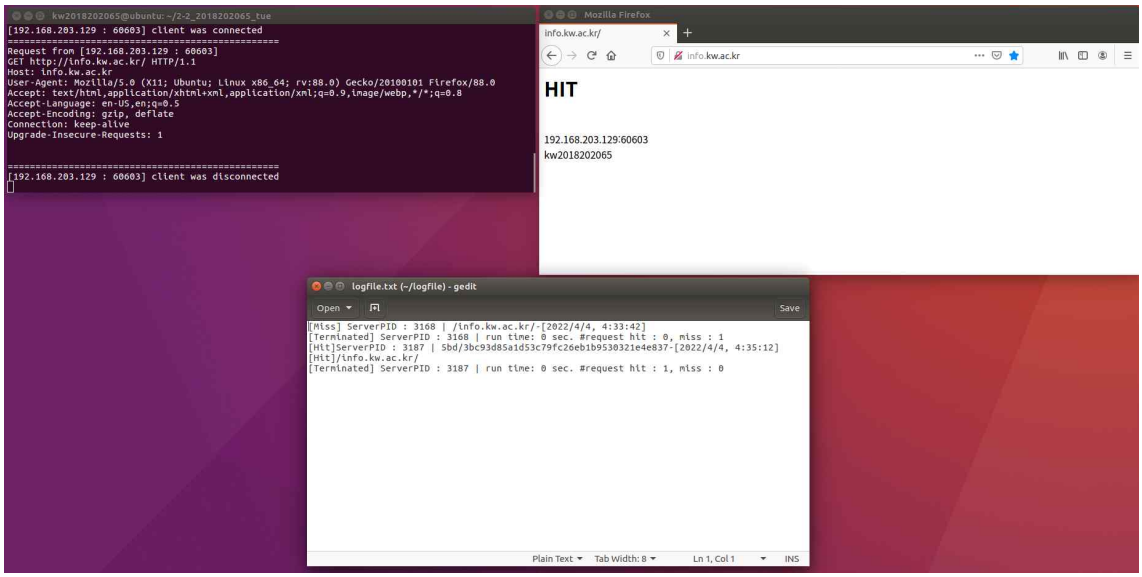
```
{
        tok = strtok(NULL, "/");
        tok = strtok(NULL, " ");
        strcpy(url, tok);
        int subprocess_misscount = 0;
        int subprocess_hitcount = 0;
        int check = sub_server_processing_helper(url);
        // 추출한 URL을 과제 1-2의 main함수였던 하지만 현재는 sub_server_processing_helper 함수에게 인자로 넘겨줌
        if (check> 0 리퀘스트에 대해 hit인 경우)
        {
                //클라이언트에게 hit을 전달함
                subprocess_hitcount++;
                sprintf(response_message,
                        "<h1>HIT</h1><br>"
                        "%s:%d<br>"
                "kw2018202065",    inet_ntoa(inet_client_address),
                client_addr.sin_port);
                sprintf(response_header,
                        "HTTP/1.0 200 OK₩r₩n"
                        "server:2018 simple web server₩r₩n"
                        "Content - length:%lu₩r₩n"
                        "content-type:text/html₩r₩n₩r₩n",
                        strlen(response_message));
        }
        else if (check == 0 리퀘스트에 대해 miss인 경우)
        {
                //클라이언트에게 miss를 전달함
                subprocess_misscount++;
                sprintf(response_message,
                        "<h1>MISS</h1><br>"
                        "%s:%d<br>"
                "kw2018202065",    inet_ntoa(inet_client_address),
                        client_addr.sin_port);
                sprintf(response_header,
                        "HTTP/1.0 200 OK₩r₩n"
                        "server:2018 simple web server₩r₩n"
                        "Content - length:%lu₩r₩n"
                        "content-type:text/html₩r₩n₩r₩n",
                strlen(response_message));
        }
```

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

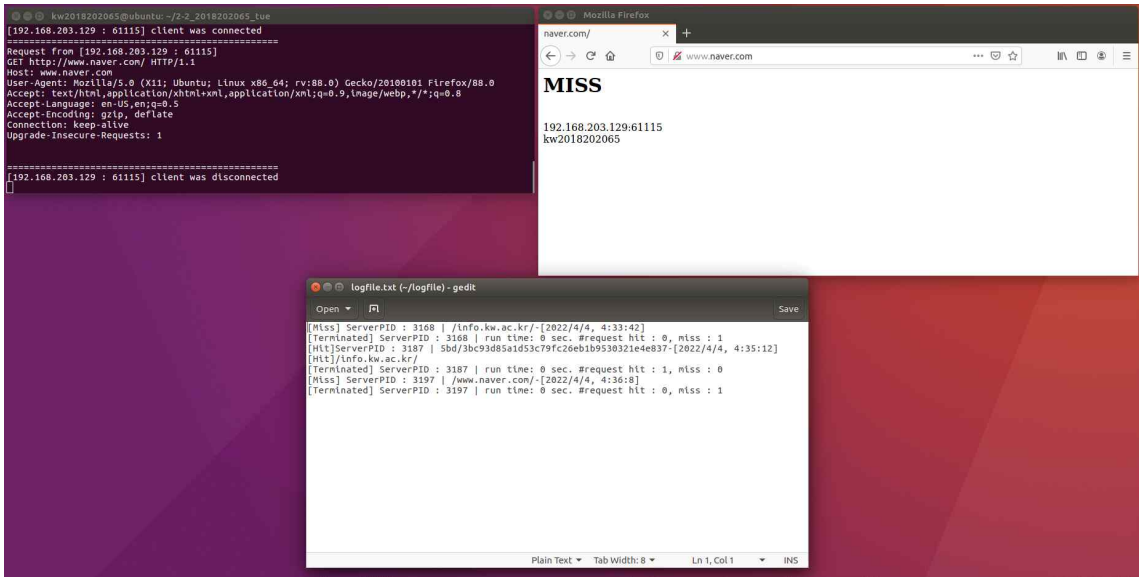
-결과 화면



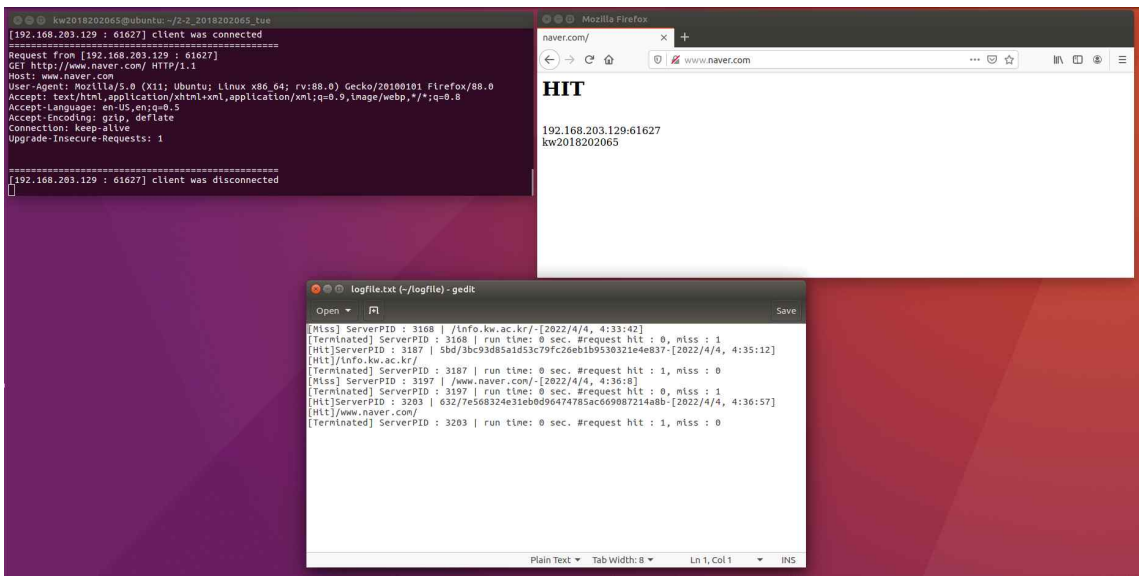
브라우저가 info.kw.ac.kr을 처음 접속할 때 miss가 결과로 나오고 클라이언트의 접속이 종료됨을 알 수 있음.



브라우저가 info.kw.ac.kr을 두번째로 접속할 때 HIT가 결과로 나오고 클라이언트의 접속이 종료됨을 알 수 있음.



브라우저가 www.naver.com을 처음 접속할 때 miss가 결과로 나오고 클라이언트의 접속이 종료됨을 알 수 있음.



브라우저가 www.naver.com을 두번째 접속할 때 hit의 결과가 나오고 클라이언트의 접속이 종료됨을 알 수 있음.

- 고찰

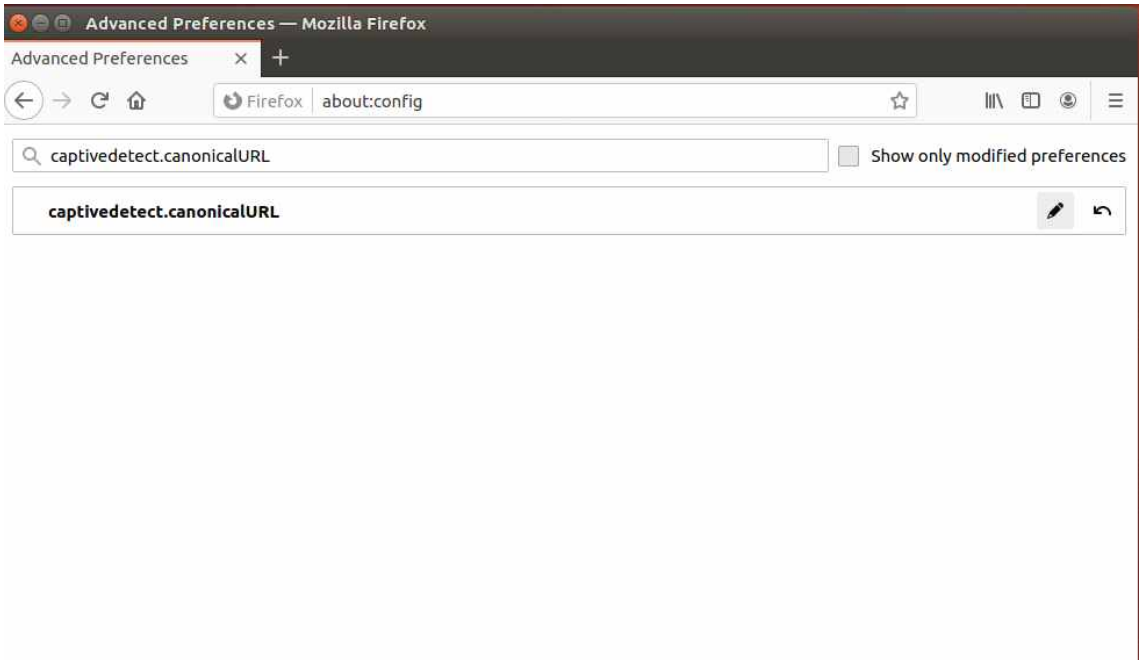
이번 과제를 진행하면서 발생한 이슈는 다음과 같다. 또한 이것에 대한 고찰을 같이 서술하고자 한다.

1. 사용자가 입력하지 않은 이상한 URL을 request하는 경우

```
[192.168.203.129 : 6803] client was connected
================================================
Request from [192.168.203.129 : 6803]
GET http://detectportal.firefox.com/success.txt HTTP/1.1
Host: detectportal.firefox.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cache-Control: no-cache
Pragma: no-cache
Connection: keep-alive


================================================
[192.168.203.129 : 6803] client was disconnected
```

먼저 detectportal.firefox.com을 request하는 경우 이는 firefox의 설정을 바꿔줌으로서 해결하였다. 이는 아래와 같다.

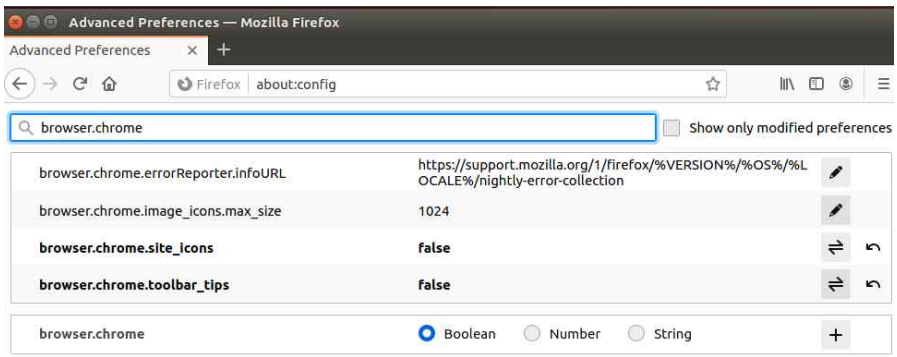
network.captive-portal-service.enabled을 true에서 false로 바꾸어 주었고 또한 이렇게 하여도 계속 입력되는 것을 확인하여

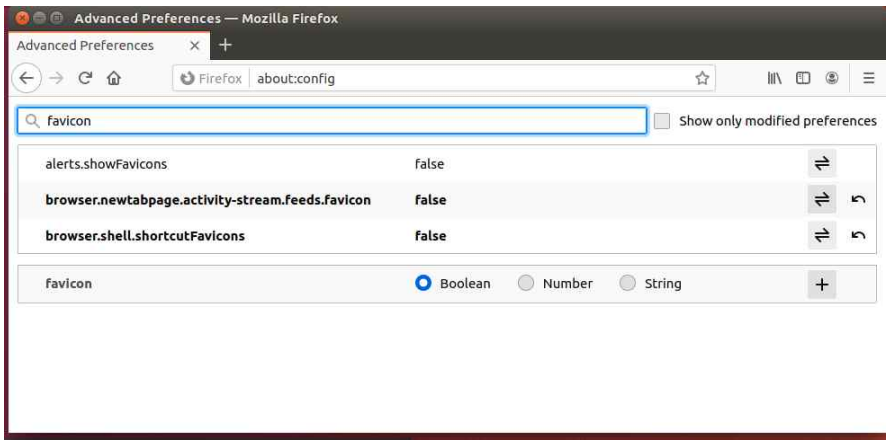


요청을 보내는 captivedetect.canonicalURL의 값
http://detectportal.firefox.com/success.txt?ipv4
http://detectportal.firefox.com/success.txt?ipv6
위의 두 개를 지워서 해결하였다.

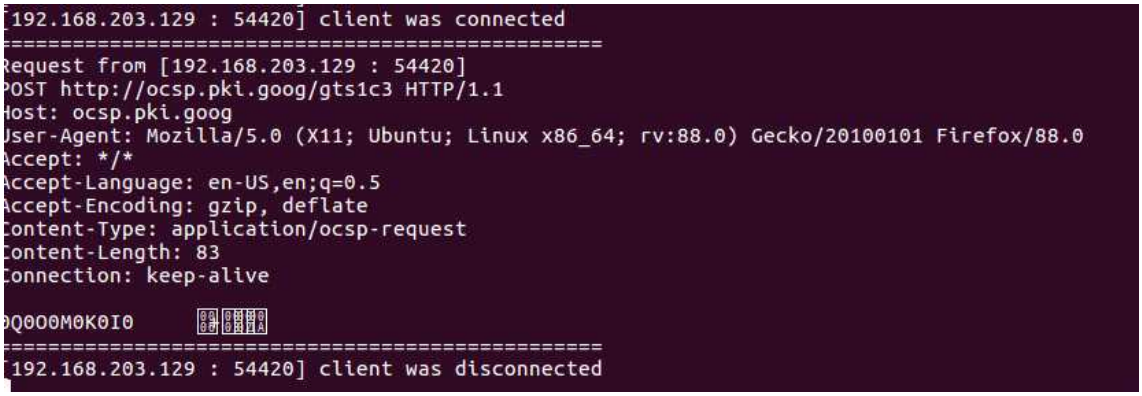
```
[192.168.203.129 : 17553] client was connected
================================================
Request from [192.168.203.129 : 17553]
GET http://www.naver.com/favicon.ico HTTP/1.1
Host: www.naver.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Fire
fox/88.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.naver.com/


================================================
[192.168.203.129 : 17553] client was disconnected
```

그다음 URL/favicon.ico가 request되는 경우 이는 firefox같이 최신 브라우저에서는 URL 왼쪽에 아이콘이 표시가 될 수 있겠끔 하는 request로



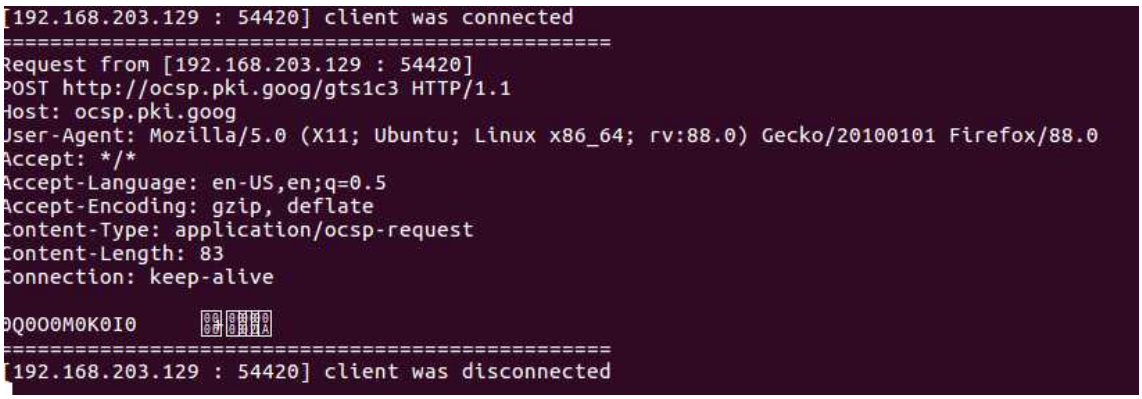


about:config에서 다음과 같이 설정을 바꿔줌으로써 해결하였다.



이러한 request도 입력이 되었지만 이는 해결하지 못해 아쉬움이 남는다. 이러한 형태는 이번 과제에서 logfile에 결과를 입력해도 무시하여 관측하였다.

2. Host URL을 추출하는 위치에 대한 이슈



이러한 request format을 보게 되면 처음 Host URL을 추출하는 부분은 Host: 부분일 것이라 생각되었지만 GET method를 통해 서버에 정보요청을 하였을 때만 cache를 생성하는 것으로 배웠기 때문에 POST, CONNECT method 등일 때는 cache를 생성하면 안 되기에 GET method가 들어왔을 때 그 뒤에 있는 URL이 더 정확한 값이라는 것을 알게 되었다.

3. 좀비 프로세스가 발생시 이를 처리하기 위한 방법에 대한 이슈

web 브라우저의 지속적인 request를 통해 fork()가 지속해서 발생할 것인데 이 중에 몇의 자식 프로세스는 좀비프로세스가 될 수 있음을 직관적으로 알 수 있고 이를 해결 하는 방법은 다음과 같은 code를 추가함으로 해결하였다.

```
if (bind(socket_fd, (struct socakaddr*)&server_addr, sizeof(server_addr)) < 0)
{
    printf("Server : Can't bind local address₩n");
    return 0;
}
listen(socket_fd, 5);
signal(SIGCHLD, (void*)handler);
```

main 함수에 있는 signal 함수에 SIGCHID 인자와 handler를 인자로 넘겨준다.

```
static void handler()
{
    pid_t pid;
    int status;
    while ((pid = waitpid(-1, &status, WNOHANG)) > 0);

}
```

main 함수 외부의 handler 함수이고 좀비 프로세스를 끊임없이 검사해 리턴 해주고 있다.

```
  }
  printf("[%s : %d] client was disconnected₩n", inet_ntoa(inet_client_address), client_addr.sin_port);
  close(client_fd);
  exit(0);
```

서브 프로세스의 종료전 exit함수를 통해 좀비 프로세스 발생을 억제하였다.

이처럼 위의 3가지 코드를 추가함으로써 좀비 프로세스 발생을 최소한으로 억제하였다.