시스템 프로그래밍 실습 Report

실습 제목: Proxy #2-3

Forward HTTP request to web server and signal handing

실습일자: 2022년 5월 02일 (월)

제출일자: 2022년 5월 18일 (수)

학 과: 컴퓨터정보공학부

담당교수: 최상호 교수님

실습분반: 목요일 7,8교시

학 번: 2018202065

성 명: 박 철 준

- Introduction

1. 제목

Proxy 2-3 Forward HTTP request to web server and signal handing

2. 목표

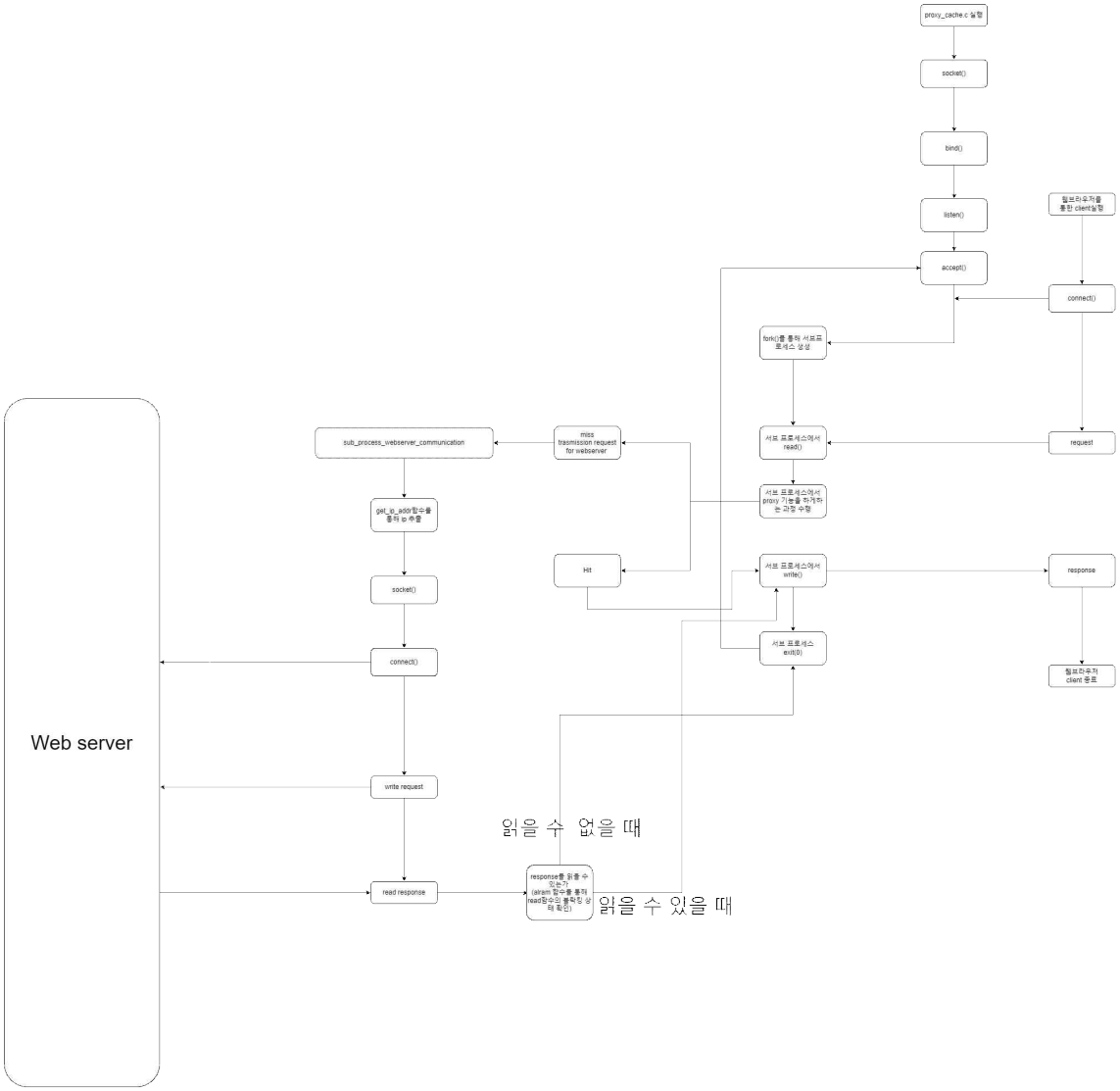
- 이번 과제에서 signal() 함수를 사용하여 SIGCHLD, SIGSALRM 시그널 처리를 해야 한다.
- Web server에 HTTP request 전송 후 10초 동안 HTTP response를 받지 못하면 화면에 "응답 없음" 메시지 출력 후 child process 종료
- 10초 이전에 web server로부터 response를 받으면 alarm 해제

3. 실험 방법 고안

위 과제에서 원하는 목표를 이루기 위해 고안한 HTTP response를 10초 동안 받지 못하게 하는 방법은 프로그램 실행 후 request를 보낸 후 인터넷을 끊어서 read를 통해 response를 받지 못하게 하는 방법이다.

● 실습 구현

-Flow Chart



-Pseudo code

이번 과제의 Pesudo code는 이전 과제에서 설명한 것은 제외하고 수정된 main 함수와 그리고 새로 구현한 get_ip_add 함수 signal handler, sub_process_webserver_communication 함수를 설명하고자 한다. 이전 과제와 달라진 main함수의 부분은 단순히 hit miss판별을 하는 기능에서 더 나아가 서브프로세스 즉 proxy 서버가 웹 서버와 통신을 할 수 있게 해주는 sub_process_webserver_communication 함수를 실행하였다는 것이다. 또한 웹 서버와의 통신을 할 수있게 도와주는 get_ip_add 함수를 이용한 모습을 볼 수 있으며 signal handler를 통해 이번 과제의 목적인 signal 처리에 대해 그 알고리즘을 알 수 있다.

main 함수

```
int main()
{

        // 목적: 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 ((socket_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
        {
                printf("server : Can't open stream socket₩n");
                //if server : Can't open stream socket
                return 0;
        }
        setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
        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 (bind(socket_fd, (struct socakaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                printf("Server : Can't bind local address₩n");
```

```
            //if Can't bind local address
            return 0;
}
listen(socket_fd, 5);
signal(SIGCHLD, (void*)handler); // 차일드 프로세스의 종료를 다루기 위해
signal(SIGALRM, (void*)handler); // response 응답 없음을 다루기 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;
            len = sizeof(client_addr);
            client_fd = accept(socket_fd, (struct sockaddr*)&client_addr, &len);
            if (client_fd < 0)
            {
                        printf("Server : accept failed₩n");
                        //if server accept failed
                        return 0;
            }
            pid = fork();
            if (pid == -1)
            {
                        //fail to fork()
                        close(client_fd);
                        close(socket_fd);
                        continue;
            }
            if (pid == 0)
            {
                        //차일드 프로세스 즉 서브 프로세스의 실행이다.
                        time_t start, end;
                        time(&start); //start to record program start time
                        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);
puts("=======================
==========================");
tok = strtok(tmp, " "); //GET 메소드인지 판별하기 위한 과정
strcpy(method, tok);
if (strcmp(method, "GET") == 0)// Get 메소드의 request일 때
URL을 통해 cache생성
{
        tok = strtok(NULL, "/");
        tok = strtok(NULL, " ");
        strcpy(url, tok);
        remove_first_char(url);
        remove_first_char(tok);
        tok = strtok(tok, "/");
        /*
        if (strcmp(tok, "detectportal.firefox.com")==0)
        {
                //테스트를 위한 예외처리

                //printf("[%s : %d] client was disconnected₩n",
                inet_ntoa(inet_client_address),
                client_addr.sin_port);
                close(client_fd);
                exit(0);
        }
        */
        int subprocess_misscount = 0;
        int subprocess_hitcount = 0;
        int check = sub_server_processing_helper(url);// 추출한
        URL을 과제 1-2의 main함수였지만
        현재는 sub_server_processing_helper 함수에게 인자로 넘
        겨줌
        if (check > 0)
        {
```

```
                //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)
        {
                //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));
                if      (sub_process_webserver_communication(tok,
                                buf) < 0)//웹서버와 통신을
                                 하는 함수!!!!
                {
                                //sub process
                                fails webserver comunication
                                in read or connect
                                function and get ip addr
                }
```

```
                                        }
                                        write(client_fd, response_header,
                                        strlen(response_header));
                                        write(client_fd, response_message,
                                        strlen(response_message));
                                        time(&end);
                                        int result = 0;
                                        result = (int)(end - start);
                                        bye(subprocess_hitcount, subprocess_misscount, result);//
                                                서브 프로세스의 종료문을 출력하기위한 함수
                                }
                                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);
                return 0;
}
```

- get_ip_addr함수

```
char* get_ip_addr(char* addr)
{
	//올바른 URL입력시 ip를 가져오는 함수
	struct hostent* hent;
	char* haddr=NULL;
	int len = strlen(addr);

	if ((hent = (struct hostent*)gethostbyname(addr)) != NULL)
	{
		haddr = inet_ntoa(*((struct in_addr*)hent->h_addr_list[0]));

	}

	return haddr;
}
```

- 시그널 handler

```
static void handler(int sig)
{
	//시그널 처리를 위한 함수
	if (sig == SIGCHLD)
	{
		//차일드 프로세스가 종료되었을 때

		pid_t pid;
		int status;
		while ((pid = waitpid(-1, &status, WNOHANG)) > 0);

	}
```

```
        if (sig == SIGALRM)
        {
                //no response로 알람이 울렸을 때
                printf("================
                No Responese=========================₩n");
        }

}
```

- sub_process_webserver_communication 함수

```
int sub_process_webserver_communication(char * host_url, char * request)
{
        // 차일드 프로세스가 서버와 통신을 할 수 있게 하는 함수이다.
        // 차일드 프로세스 즉 프록시 서버가 클라이언트가 되어 웹서버에 컨넥트
        // 를 요청하고 request를 보내고 웹서버로 부터 requst를 받아냄
        int server_fd, len;
        int error = 0;
        char buf[BUFFSIZE] = { 0, };
        struct sockaddr_in server_addr;
        char* haddr = NULL;
        haddr=get_ip_addr(host_url);
        struct sigaction act, oact;
        act.sa_handler = handler;
        sigemptyset(&act.sa_mask);
        act.sa_flags &= ~SA_RESTART;
        sigaction(SIGALRM, &act, &oact);
        if (haddr == NULL)
        {
                //fail to accept address for webserver.
                return -1;
        }

        if ((server_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
        {
                //can't create socket.
                return -1;

        }
```

```
	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(80);

	if (connect(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
	{
		//can't connect.
		close(server_fd);
		return -1;

	}
	else
	{
		//connect된걸 확인하고 랜선을 끊기 위해 출력문 사용
		printf("connect success.₩n");

	}

	write(server_fd, request, strlen(request));
	alarm(10);

	if (read(server_fd, buf, BUFFSIZE) < 0)
	{
		//read error.
		close(server_fd);
		return -1;
	}
	else
	{
		alarm(0);
		//can read.
	}
	close(server_fd);
	//함수 종료
	return 0;
}
```

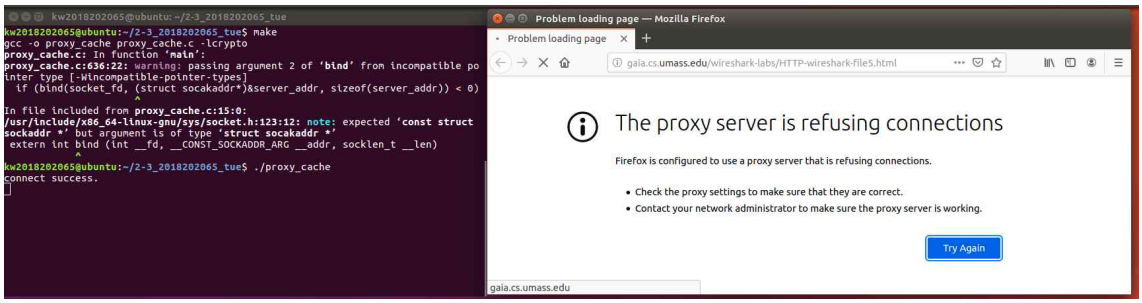
● 실습 결과

-실습 방법

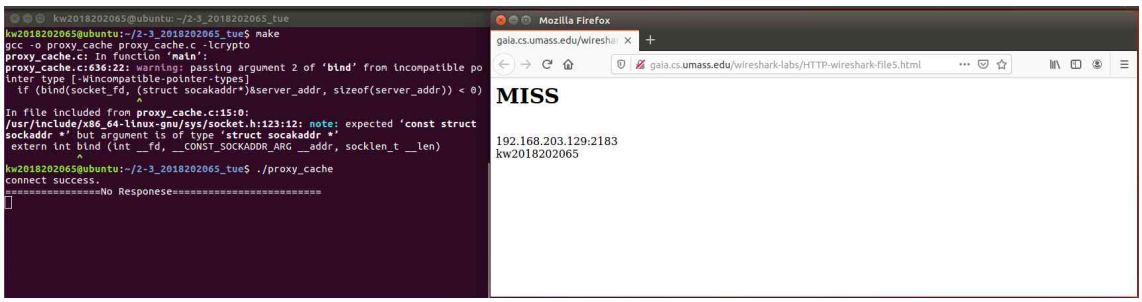
먼저 miss일 경우만 웹서버와 통신을 함으로 cache에 없는 url을 먼저 입력 한다. 이후 프록시 서버는 socket을 실행하고 웹 서버에 connect를 할 것이며 connect를 성공하면 connect success문구가 출력되고 프록시 서버는 write를 통해 웹 브라우져의 request를 웹 서버로 보낼 것인데 이후 read를 통해 response를 받기 전에 타이밍을 맞추어 인터넷 연결을 끊게 되고 이때 read를 하게 되면 프록시 서버는 웹 서버의 response를 받지 못하게 되고 이로 인해 블락킹 상태에 도립한다. 이때 블락킹 상태가 10초가 지나가면 설정해 두었던 alarm 함수로 인해 signal 핸들러가 실행 될 것이고 핸들러는 터미널에 "no response"를 출력하고

read 함수는 오류를 반한하고 웹브라우져에 miss화면을 출력 서브 프로세스를 종료하게 된다. 이를 사진을 통해 보면 아래와 같다.

-결과 화면



처음 컴파일 후 proxy_cache를 실행하고 웹브라우져에서 http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html 을 입력후 터미널에서 connect succes가 뜬 경우 바로 인터넷 연결을 해제 하면 alarm 함수를 통해 시간을 카운트하게 되는 상태이다.



이후 10초가 지나면 "No Response" 출력문과 함께 서브 프로세스는 종료되어 새로운 웹 브라우져와의 통신을 기다리는 상태가 됨을 터미널을 통해 알 수 있다. 또한 웹 브라우져 화면은 Miss여야지만 서버와 통신함을 알 수 있기에 이러한 출력을 나타낼 수 있게 하였다.

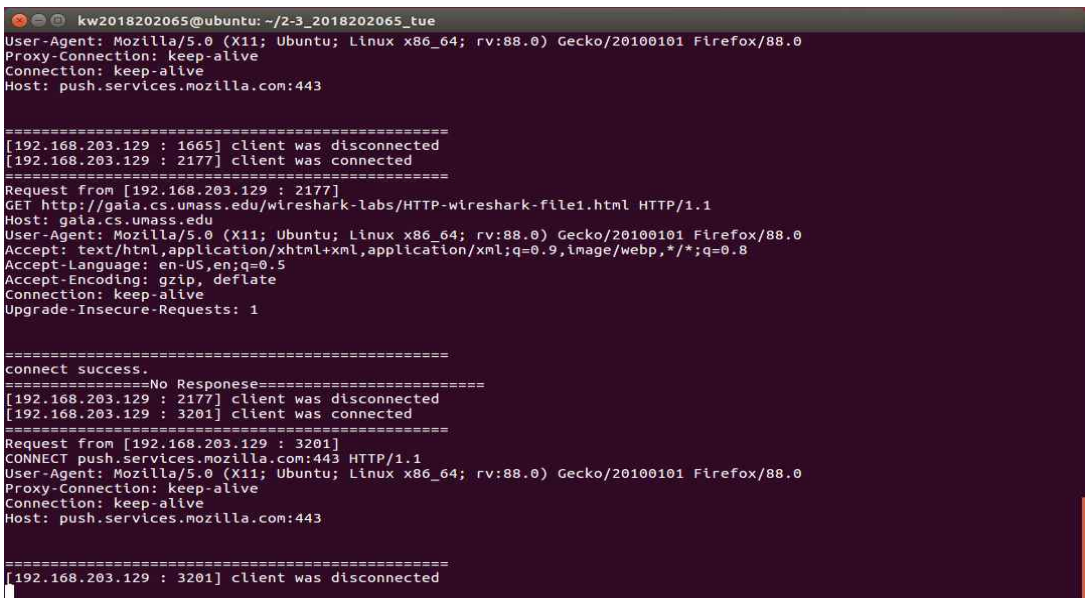
● 고찰

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

1. response를 받지 못하는 상태에 대한 정의

먼저 이번 과제의 목표인 response를 받지 못하는 상태는 어떠한 상태일 것인가에 대해 고민하여 보았고 예를들어 만약 호스트 URL을 파싱하여 IP를 얻어오는 과정에 있어 IP를 가져올 없는 URL인 경우(www.dfdsafadsf.com)같은 경우 이러한 경우도 response를 받지 못하는 경우 일지도 모른다고 생각했고 또한 connect를 하기도 전에 인터넷 연결이 끊어진 경우 connect를 통해 웹서버와 통신을 못 하게 되니 이러한 경우도 웹서버에서 response를 받지 못하는 것이 아닌가에 대해 생각해 보았다. 하지만 가장 먼저의 경우 올바르지 않은 .URL을 파싱하여 IP를 얻어오는 과정을 실패한 경우에는 이번 과제의 목적인 response를 받지 못하는 경우가 아닌 서버의 ip를 받지 못하는 경우가 더욱 맞다 생각되어 이러한 생각을 정리할 수 있었고 두 번째 connect를 하기 전 인터넷 연결을 끊어 connect를 하지 못하는 경우에 대해서는 response를 받지 못한다기보다는 서버와 연결을 할 수 없는 경우에 더 가깝다고 생각되어 졌다. 고로 서버에게 response를 받지 못하는 상황에 대한 정의에 대해 내가 생각한 결론은 올바른 URL을 입력받은 후 connect를 통해 서버와 통신을 할 수 있는 상태에서 write 함수를 통해 request를 보내고 웹서버에서는 이러한 request에 대한 response를 생산 하여 전달 하여 주는 상태에서 proxy 서버에서 인터넷이 끊겨 read를 하여도 웹서의 response를 받지 못하는 경우라는 결론에 이르게 되었다. 이는 코드로 적용하여 이번 과제를 구현하는 데 있어 프로그램 흐름과 알고리즘을 구현하는 데 도움을 주었다.

2. alarm을 받는 10초 동안에 다른 웹 브라우져(사용자가 원해서 접속하는 웹브라우져가 아닌 시스템상에서 자동으로 접속되어지는 웹 부라우져)가 프록시 서버에 연결되어 터미널창을 어지럽히게 되는데 이번 과제의 결과화면에 있어서는 최상의 결과값이 출력되어야 하기 때문에 최대한 이러한 웹 브라우져의 접속을 안뜨게 하였지만 만약 다른 프록시 서버의 관리자가 프록시 서버가 동작하는 터미널을 봤을 경우 다음과 같은 화면을 볼 수 있다.



보기에 심각한 오류가 난 것으로 보여지지만 전혀 오류가 아니다. alarm을 통한 10초의 시간 동안 웹 브라우져는 통신을 할 수 있어야 하고 또 다른 웹 브라우져와 connect가 된다면 또 다른 child 프로세스가 생성될 것이고 Proxy 서버의 기능을 할 것이다. 이는 또한 정상적인 결과가 끝나게 되면 정상 종료 할 것이고 이는 다중 차일드 프로세스가 종료되어 좀비 프로세스로 남게 되는 문제 또한 SIGCHLD 시그널 처리로 인해 없애 버렸기 때문에 문제가 될 상황은 발 생하지 않는다. 또한 10초가 끝나면 no response를 출력 후 해당 차일드 프로세스는 종료될 것이고 이후 다시 다른 차일드 프로세스들이 생성되어 화면에 출력될 것이다.