시스템 프로그래밍 실습 Report

실습 제목: Proxy #2-4

1. Forward HTTP request to web server and signal handling
2. Add cache and log to proxy server

실습일자: 2022년 5월 16일 (월)

제출일자: 2022년 5월 25일 (수)

학 과: 컴퓨터정보공학부

담당교수: 최상호 교수님

실습분반: 목요일 7,8교시

학 번: 2018202065

성 명: 박 철 준

- Introduction

1. 제목

Proxy 2-4

1. Forward HTTP request to web server and signal handling
2. Add cache and log to proxy server

2. 목표

- 이번 과제에서는 miss일 경우 각 URL에 해당하는 cache파일 내에 HTTP response를 저장하여 한다.

-SIGINT(Ctrl+C)를 입력받게 되면 현재까지의 동작한 프로그램 시간과 현재까지 발생한 서브 프로세스의 개수를 기록하는 종료문을 출력하여야 한다.

- Hit일 경우는 각 URL의 해당하는 cache파일 내에서 HTTP responsemf 가져와야한다.

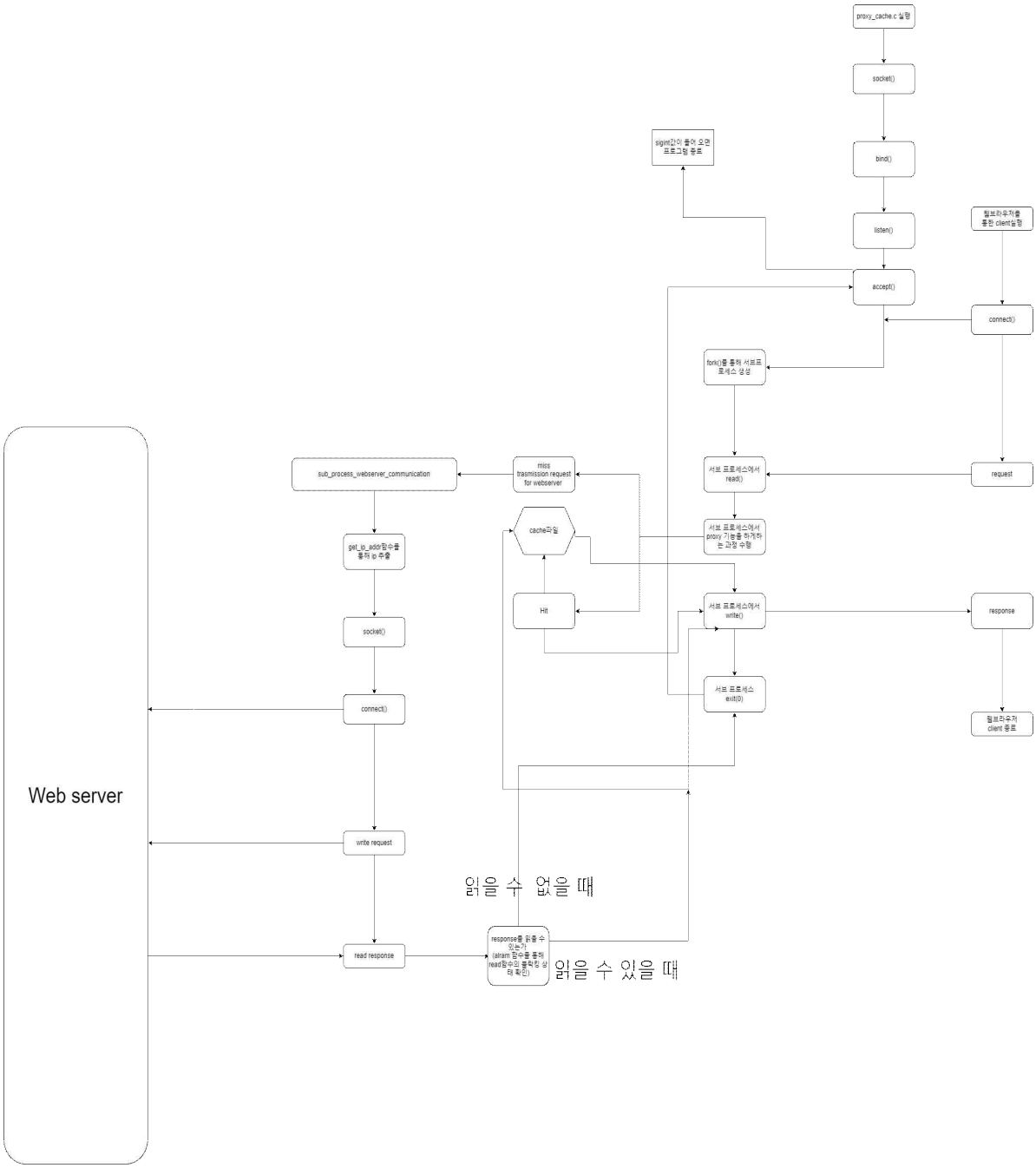
3. 실험 방법

2-3에서 구현한 웹 서버와의 소켓 통신을 이용하여 request 전달과 response 응답을 처리하는 새로운 코드를 작성한다. 또한 2-3에서 구현한 시그널 핸들러에 SIGINT도 처리할 수 있는 방법으로 코딩을 하여 구현한다.

● 실습 구현

-Flow Chart

2-3 과제에서의 플로 차트와 비교하면 miss일 경우 각 URL에 해당하는 cache 파일 내에 HTTP response를 저장하는 과정을 추가하였다. 또한 HIT일 경우 이를 사용하여 브라우저에 전달하는 과정을 추가하였다. 마지막으로 SIGINT(Ctrl+C)의 입력이 들어올 경우 프로그램을 종료하는 부분도 추가하였다.



-Pseudo code

이번 과제의 Pesudo code는 이전 과제에서 설명한 것은 제외하고 수정된 main 함수와 그리고 수정된 sub_process_webserver_communication 함수 그리고 수정된 시그널 핸들러 함수 또 구현한 read_with_timeout 함수 bye_program 함수를 설명하고자 한다. 이전 과제와 달라진 sub_process_webserver_communication 함수의 부분은 리턴값을 서버의 파일 디스크립터 값로 받아 main함수에서 사용할 수 있도록 변경하였고 main함수의 수정 부분은 Hit Miss를 판별한 뒤 웹 서버와 소켓 통신 후 http request 전달 및 수신을 하고 이때 수신 한 메시지를 miss일 경우임으로 캐시에 저장하고 브라우저에 전달 한다.

main 함수

코드가 긴 관계로 바뀐 부분만 설명하고자 한다.

```
if (check > 0)
{
	//Hit일 경우
	//클라이언트에게 hit을 전달함
	subprocess_hitcount++;
	char dire[100];
	char HS_URL[100];
	char first_HS_URL[4];
	char second_HS_URL[100];
	sha1_hash(hash_passing_url, HS_URL);//SHA1을 하기위해 입력받은 문자열과 아웃풋을 저장할 변수를 인자로하여 함수 실행.
	int i;
	for (i = 0; HS_URL[i] != '₩0'; i++)
	{
		if (i < 3)
		{
			//디렉토리명을 정해주기 위해 해쉬된 URL을 분해한다.
			first_HS_URL[i] = HS_URL[i];
			first_HS_URL[i + 1] = '₩0';
		}
		else
		{
			//파일명을 정해주기 위해 해쉬된 URL을 분해한다.
			second_HS_URL[i - 3] = HS_URL[i];
			second_HS_URL[i - 3 + 1] = '₩0';

		}
	}
	getHomeDir(dire);
```

```
        strcat(dire, "/cache/");
        strcat(dire, first_HS_URL);
        strcat(dire, "/");
        strcat(dire, second_HS_URL); //존재하는 디렉토리에 파일을 생성 하기위해 파일 명을 저장한다.
        int cache_file_fd = 0;
        if ((cache_file_fd = open(dire, O_RDWR | O_APPEND,0777)) >=0) //생성된 캐시파일을 열어줌
        {
                //디렉토리 내부에 해쉬된 파일이 있는지 확인을 위한 출력문 성공일 경우
                //printf("open succes₩n");
        }
        else
        {
                //실패일 경우
                printf("open error₩n");
                close(cache_file_fd);
        }
        int read_num = 0;
        while ((read_num = read(cache_file_fd, buf, BUFFSIZE)) > 0)//파일에서 읽어 온 값이 0포함 작을 때까지 즉 읽을 값이 없거나 오류일 때까지
        {
                //recieve HTTP
                //read의 반환값을 출력하기 위한 출력문
                //printf("read_num is %d₩n", read_num);
                write(client_fd, buf, read_num);
                bzero(buf, BUFFSIZE);
        }
        close(cache_file_fd);
}
```

```
else if (check == 0)
{
        //클라이언트에게 miss를 전달함
        subprocess_misscount++;
        int server_fd ;
        if ((server_fd=sub_process_webserver_communication(getip_passing_url, buf)) <
        0)//웹서버와 통신을 하는 함수!!!!
        {
                //웹서버와 통신의 실패를 출력하는 함수 //printf("sub process fails
                webserver comunication in read or connect function and get ip
                addr₩n");
        }
        else
        {
                char dire[100];
                char HS_URL[100];
                char first_HS_URL[4];
                char second_HS_URL[100];
                sha1_hash(hash_passing_url, HS_URL);//SHA1을 하기위해 입력받은 문자열
                과 아웃풋을 저장할 변수를 인자로하여 함수 실행.
                int i;
                for (i = 0; HS_URL[i] != '₩0'; i++)
                {
                        if (i < 3)
                        {
                                //디렉토리명을 정해주기 위해 해쉬된 URL을 분해한다.
                                first_HS_URL[i] = HS_URL[i];
                                first_HS_URL[i + 1] = '₩0';
                        }

                        else
                        {
                                //파일명을 정해주기 위해 해쉬된 URL을 분해한다.
                                second_HS_URL[i - 3] = HS_URL[i];
                                second_HS_URL[i - 3 + 1] = '₩0';
                        }
                }
                getHomeDir(dire);
                strcat(dire, "/cache/");
                strcat(dire, first_HS_URL);
                strcat(dire, "/");
```

```
strcat(dire, second_HS_URL);
FILE* pFile =0;
int cache_file_fd = 0;
if ((cache_file_fd = open(dire, O_RDWR | O_APPEND,0777)) >= 0)
{
        //캐시파일의 파일 오픈이 되는지 확인을 위한 출력문
        //printf("open succes₩n");
        close(cache_file_fd);
}
else
{
        printf("open error₩n");
        close(cache_file_fd);
}
//핸들러가 실행할 때 대기상태를 벗어나기위한 부분
struct sigaction act, oact;
act.sa_handler = handler;
sigemptyset(&act.sa_mask);
act.sa_flags &= ~SA_RESTART;
sigaction(SIGALRM, &act, &oact);

write(server_fd, buf, strlen(buf));

alarm(15);//알람 15초 설정

bzero(buf, BUFFSIZE);
int read_num = 0;
int read_num_sum=0;
int check_num = 0;
int Content_Length_int = 0;
int Content_Length_before_entity_body = 0;
bool chunck_check = false;
int newline_count = 0;


while ((read_num= read_with_timeout(server_fd, buf, BUFFSIZE,5000))>0)
{
        //recieve HTTP
        alarm(0);// read값을 반환하여 while문을 실행 시킨경우 알람해제

        read_num_sum += read_num;
```

```
check_num += 1;

if (check_num == 1)
{
	char tmp1[BUFFSIZE] = { 0, };
	char tmp2[BUFFSIZE] = { 0, };
	char* tok1 = NULL;
	char* tok2 = NULL;
	strcpy(tmp1, buf);
	strcpy(tmp2, buf);
	tok1 = strtok(tmp1, "₩n");
	Content_Length_before_entity_body += strlen(tok1);
	newline_count++;
	while (strcmp(tok1, "₩r") != 0)
	{
		tok1 = strtok(NULL, "₩n");
		Content_Length_before_entity_body +=
		strlen(tok1);
		newline_count++;
	}
	tok2 = strtok(tmp2, " ");
	while(strcmp(tok2, "Content-Length:") != 0)//
	Content-Length: 글자 탐색
	{
		if (strcmp(tok2, "Transfer-Encoding:") ==
		0)//Transfer-Encoding: 글자 탐색
		{
			chunck_check = true;
			break;
		}
		tok2 = strtok(NULL, "₩n");
		tok2 = strtok(NULL, " ");
	}
	tok2 = strtok(NULL, "₩r");
	char* Content_Length_char = NULL;
	Content_Length_char = tok2;
	Content_Length_int = atoi(Content_Length_char);
	//컨텐츠 길이의 숫자가 알맞게 추출되었는지 확인을 위
	한 출력문 //printf("Content_Length_int is %d ₩n",
	Content_Length_int);
	//entity body 전까지의 리스폰스 메세지의 길이를 알맞
```

```
                게 계산했는 지 확인을 위한 출력문
                //printf("Content_Length_before_entity_body is %d ₩n",
                Content_Length_before_entity_body);

        }
        //read의 반환값을 출력을 위한 출력문 //printf("read_num is
        %d₩n", read_num);
        write(client_fd, buf, read_num);
        pFile = fopen(dire,"ab");
        fwrite(buf,1, read_num,pFile);
        fclose(pFile);
        bzero(buf, BUFFSIZE);
        if ((Content_Length_int + Content_Length_before_entity_body+
        newline_count <= read_num_sum)&&(chunck_check==false))
        {

                /* 추출한 read_num의 길이의 합이 엔티티 바디 길이 포
                함 헤더 길이 포함과 같은지 확인
                printf("newline_count is %d₩n", newline_count);
                printf("Content_Length_int is %d ₩n",
                Content_Length_int);
                printf("Content_Length_before_entity_body is %d ₩n",
                Content_Length_before_entity_body);
                printf("read_num_sum is %d₩n", read_num_sum);
                printf("Content_Length_int +
                Content_Length_before_entity_body+ newline_count is
                %d ₩n", newline_count+Content_Length_int +
                Content_Length_before_entity_body);
                */
                alarm(0);// content length를 포함하여 content length를
                알수 있을 때 read_with_time함수가 기다리는 값보다 더
                빨리 종료 되게 함으로
                // 알람을 0으로 만듬
                close(server_fd);
        }
        alarm(15);// read를 다시실행 할 경우를 대비하여 알람 15초 설정
        이는 while값을 벗어나면 해제 된다.
}
//while 값을 종료한 지점의 read_num 값이 0인지 확인 //printf("while
end point read_num is %d₩n", read_num);
if (read_num == 0)
```

```
                {
                        //read가 안전하게 끝나면 알람 해제
                        alarm(0);
                        close(server_fd);
                }
                else if (read_num == -1)
                {
                        //read를 실패함을 알려주는 출력문 //printf("while end point
                        read_num is %d₩n", read_num);
                        close(server_fd);
                }
                close(server_fd);
        }
}
```

sub_process_webserver_communication() 함수

sub_process_webserver_communication() 함수의 경우 2-3 과제에서는 웹서버에 브라우저의 request를 전달하고 브라우저에 response를 전달하는 과정까지 코드작성하였지만 이번 과제에 있어서는 웹서버의 파일 디스크립터만 반환하는 방식으로 변경하였다.

```
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 웹서버 소켓의 파일 디스트립터값을 반환한다.
	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);

	if (haddr == NULL)//URL에서 아이피를 가져올 수 없는 경우
	{
		//fail to accept address for webserver.
		return -1;
	}

	if ((server_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
	//웹서버 파일디스크립터를 소켓을 설정해 저장
	{
		//웹서버 소켓을 만들 수  없는 경우
		//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)
	//웹서버와 소켓 컨넥트
	{
		//can't connect.
		close(server_fd);
		return -1;
	}
```

```
        else
        {
                //컨넥트 성공
                //connect된걸 확인하고 랜선을 끊기 위해
                //connect된것을 확인하는 출력문 //printf("connect success.\n");
                return server_fd;
        }
        return server_fd;
}
```

시그널 핸들러 함수
이전 과제의 핸들러 함수와 다른 점은 SIGINT 시그널에 대한 처리도 추가했다는 점이다. 이는 아래와 같다.

```
static void handler(int sig)
{
	//핸들러 함수
	if (sig == SIGCHLD)
	{
		//차일드 프로세스가 종료되었을 때
		//차일드 프로세스가 잘 죽었는지 확인을 위한 출력문
		//printf("================child die=========================\n");
		pid_t pid;
		int status;
		while ((pid = waitpid(-1, &status, WNOHANG)) > 0);
	}

	if (sig == SIGALRM)
	{
		printf("================No Responese=========================\n");
		//핸들러에서 시그알람일 때 만약 서브프로세스를 종료한다면 사용
		//bye(0,1,10);
		//핸들러에서 시그알람일 때 만약 서브프로세스를 종료한다면 사용//exit(0);
	}

	if (sig ==SIGINT)
	{
		long detect_processid = (long)getpid();//현재 프로세스 아이디를 저장
		//현재 프로세스 아이디를 출력하는 출력문 //printf("detect_processid: %ld
		\n", detect_processid);
		if (detect_processid != global_processid)// 전역변수로 저장한 부모 프로세
		스 아이디와 비교 해서 다르면
		{
			//자식 프로세스임으로 종료!
			exit(0);
		}
		time(&global_end);//끝난 시간을 기록
		int global_result = 0;
		global_result = (int)(global_end - global_start);// 프로그램 동작 시간 기록
		printf("\n");
		bye_program(global_result);//프로그램 종료문 출력!
		exit(0);//부모 프로세스 즉 프로그램 종료!
	}
}
```

read_with_timeout () 함수

위 함수는 웹서버와 소켓 통신이 성공하고 나면 프록시가 보낸 request에 대한 웹서버의 response를 read함수로 받게 되는데 read의 대기상태를 예방하기 위해 새로 정의 한 타임아웃이 적용된 read함수 이다. 이는 아래와 같다.

```
int read_with_timeout(int fd, char* buf, int buf_size, int timeout_ms)
{
	//웹서버 소켓을 통해 사용하는 read를 타임아웃 설정을 하여
	//read_with_timeout으로 재정의
	int rx_len = 0;
	struct timeval timeout;
	fd_set readFds;

	// recive time out config
	// Set 1ms timeout counter
	timeout.tv_sec = 0;
	timeout.tv_usec = timeout_ms * 1000;

	FD_ZERO(&readFds);
	FD_SET(fd, &readFds);
	select(fd + 1, &readFds, NULL, NULL, &timeout);

	if (FD_ISSET(fd, &readFds))
	{
		rx_len = read(fd, buf, buf_size);
	}

	return rx_len;
}
```

bye_program() 함수
SIGINT(Ctrl+C)가 들어오면 프로그램을 종료시킬 때 출력문을 작성하는 함수로 pseudo 코드는 아래와 같다.

```
void bye_program(int global_result)
{

        // 프로그램 종료문을 형식에 맞게 입력하는 함수

        char dire[100];
        getHomeDir(dire);
        strcat(dire, "/logfile/");
        strcat(dire, "logfile.txt");
        int fd;
        fd = open(dire, O_WRONLY | O_APPEND, 0777);
        // 아래는 종료문을 형식에 맞게 입력하는 과정
        char st[100];
        memset(st, 0, 100);
        write(fd, "**SERVER** [Terminated] run time: ", strlen("**SERVER** [Terminated]
        run time: "));
        sprintf(st, "%d", global_result);
        write(fd, st, strlen(st));
        write(fd, " sec. ", strlen(" sec. "));
        write(fd, "#sub process: ", strlen("#sub process: "));
        sprintf(st, "%d", process_count);
        write(fd, st, strlen(st));
        write(fd, "₩n", strlen("₩n"));
        close(fd);
        return;
}
```

- 실습 결과

-실습 방법
Hit or Miss일 경우 브라우져에게 알맞은 데이터를 올바른 방식으로 전달 하는지 확인하는 방법
1. 가장먼저 ./proxy_cache를 통해 프록시 서버를 실행한다. 이후 로그파일이 생성되는지 확인
2. http://sslab.practice.s3-website.ap-northeast-2.amazonaws.com/을 브라우저 입력하여 miss 상태에서 캐시를 만들고 브라우저에는 웹 서버의 response출력 로그에 저장된 문구를 확인하여 miss 상태임을 확인
3. http://sslab.practice.s3-website.ap-northeast-2.amazonaws.com/bsai.html을 브라우저 입력하여 miss 상태에서 캐시를 만들고 브라우저에는 웹 서버의 response출력 로그에 저장된 문구를 확인하여 miss 상태임을 확인
4. http://neverssl.com/을 브라우저 입력하여 miss 상태에서 캐시를 만들고 브라우저에는 웹 서버의 response출력 로그에 저장된 문구를 확인하여 miss 상태임을 확인
5. http://sslab.practice.s3-website.ap-northeast-2.amazonaws.com/bsai.html을 입력하고 Hit 상태임으로 만들어진 캐시에서 데이터를 브라우저에게 전달하여 창이 제대로 뜨는 것을 확인하고 로그파일을 확인하여 Hit 상태를 확인한다.

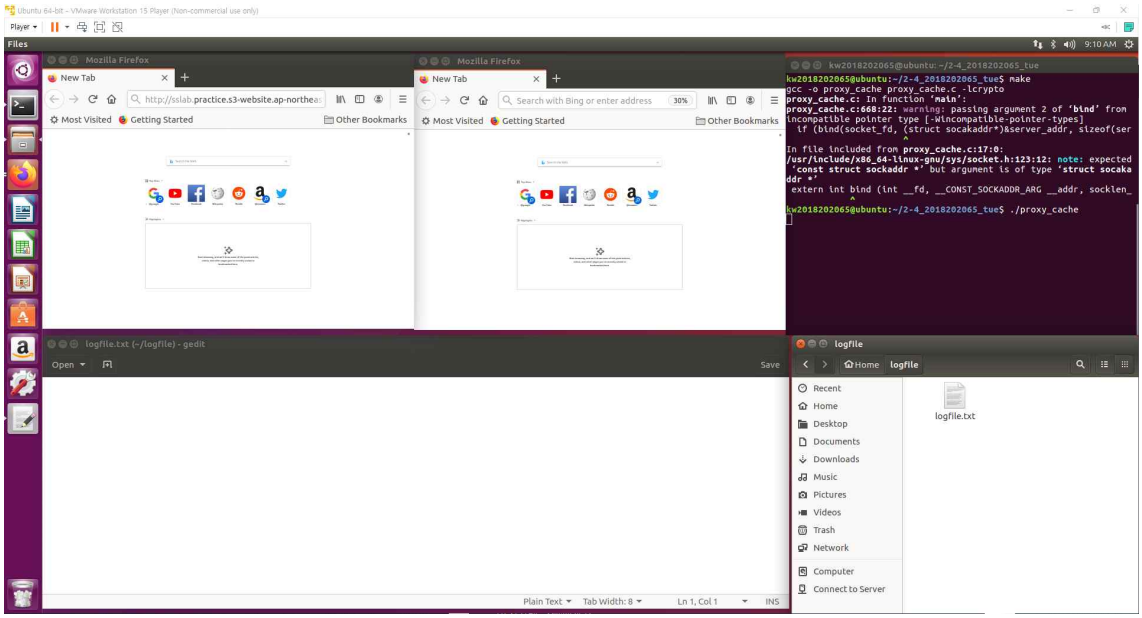
SIGINT의 요구사항을 만족하는 것을 확인하기 위한 방법
6. http://neverssl.com/을 브라우저 입력하여 miss 상태에서 캐시를 만들고 브라우저에는 웹 서버의 response출력 로그에 저장된 문구를 확인하여 miss 상태임을 확인
7. http://neverssl.com/을 입력하고 Hit상태임으로 만들어진 캐시에서 데이터를 브라우저에게 전달하여 창이 제대로 뜨는 것을 확인하고 로그파일을 확인하여 Hit 상태를 확인한다.
8. Ctrl+C를 입력하고 프로그램을 종료하고 이에 대한 종료문이 알맞게 작성되었는 지 확인한다.

예시 웹사이트 제외하고 http를 가지는 다른 웹사이트 접속하여 보기
9. 환경부 http://me.go.kr/home/web/main.do에 접속하여 잘 되는 것을 확인

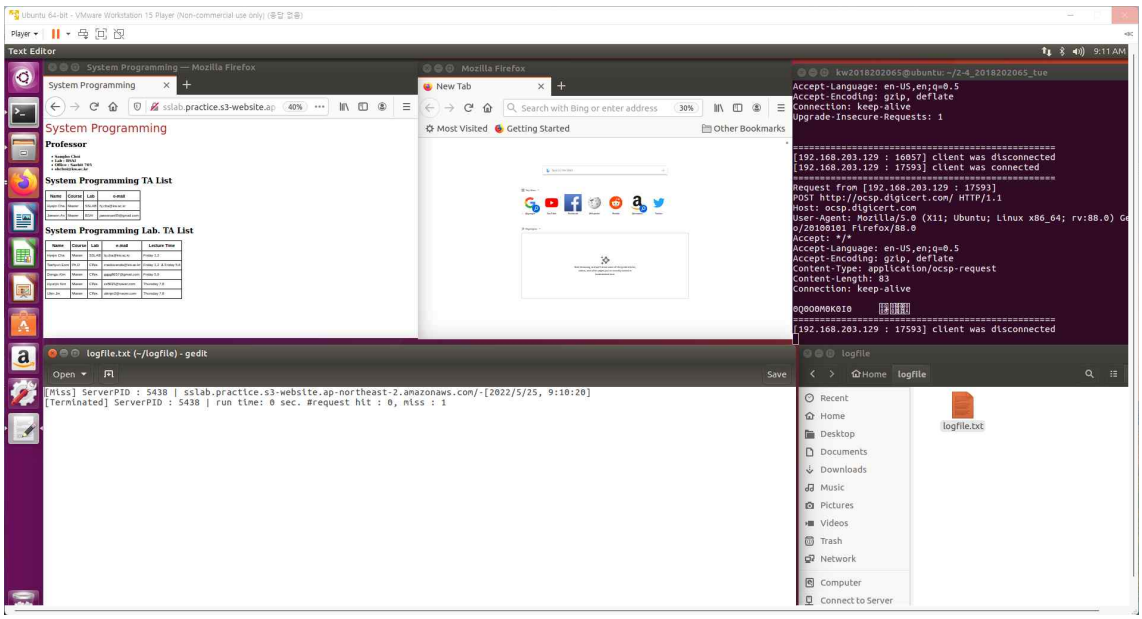
각 실습 방법에 대한 결과는 아래와 같다.

-결과 화면

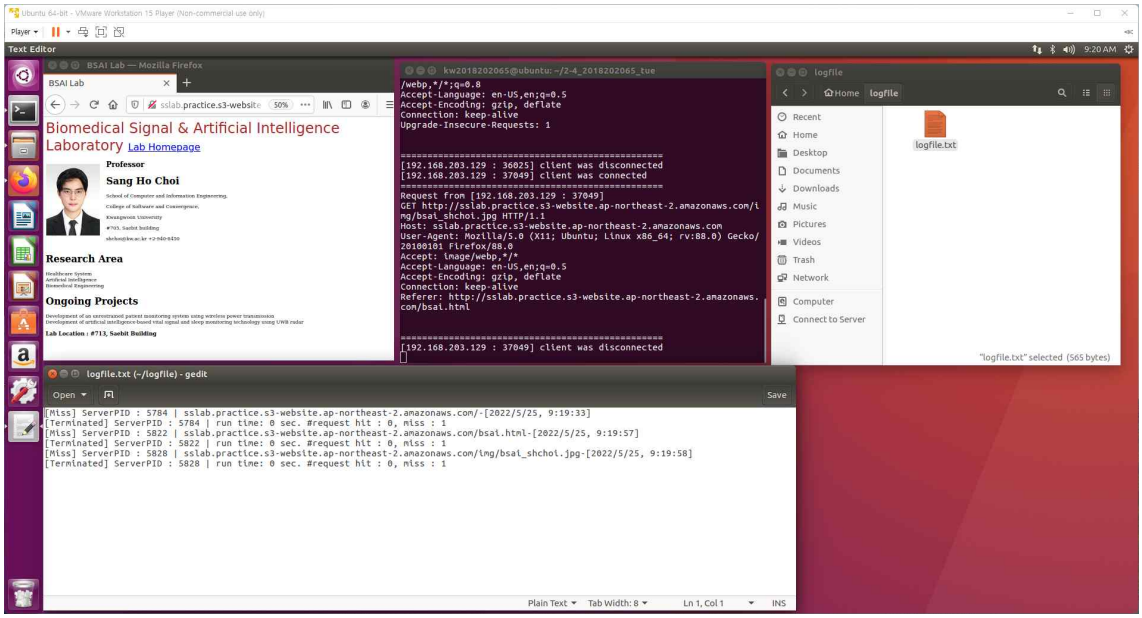
1.


kw2018202065@ubuntu:~/2-4_2018202065_tue$ make
gcc -o proxy_cache proxy_cache.c -lcrypto
proxy_cache.c: In function 'main':
proxy_cache.c:668:22: warning: passing argument 2 of 'bind' from incompatible pointer type [-Wincompatible-pointer-types]
if (bind(socket_fd, (struct sockaddr*)&server_addr, sizeof(ser
^
In file included from proxy_cache.c:17:0:
/usr/include/x86_64-linux-gnu/sys/socket.h:123:12: note: expected 'const struct sockaddr *' but argument is of type 'struct sockaddr *'
extern int bind (int __fd, __CONST_SOCKADDR_ARG __addr, socklen_
^
kw2018202065@ubuntu:~/2-4_2018202065_tue$ ./proxy_cache
logfile.txt (~/logfile) - gedit
logfile
logfile.txt

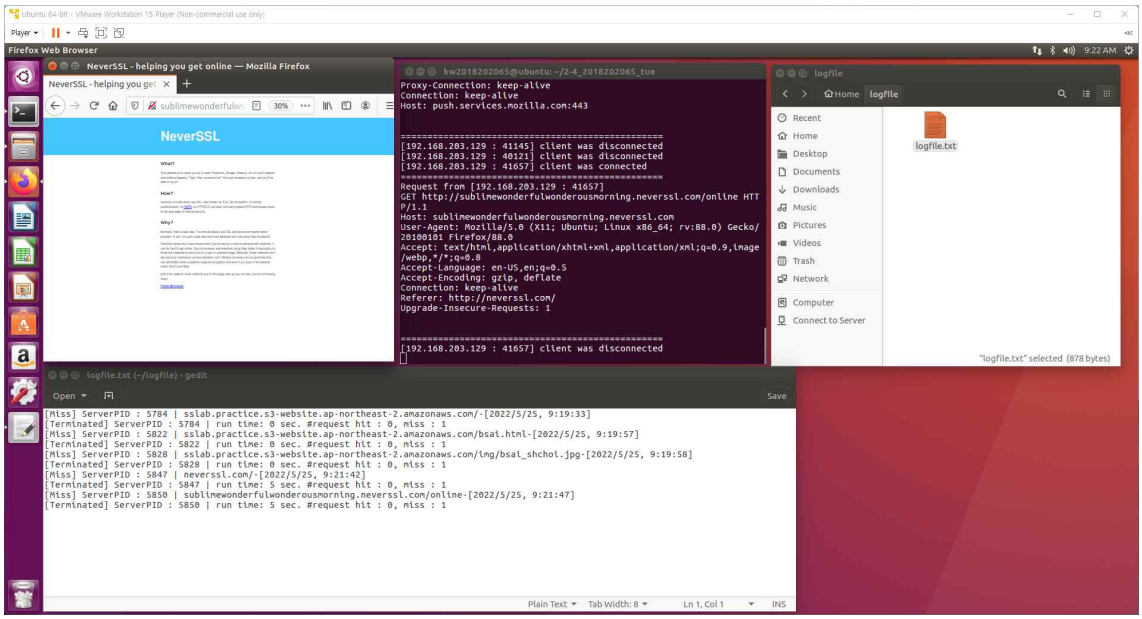
2.


System Programming
Professor
System Programming TA List
System Programming Lab. TA List
[Miss] ServerPID : 5438 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/-[2022/5/25, 9:10:20]
[Terminated] ServerPID : 5438 | run time: 0 sec. #request hit : 0, miss : 1
[192.168.203.129 : 16057] client was disconnected
[192.168.203.129 : 17593] client was connected
[192.168.203.129 : 17593] client was disconnected
logfile.txt (~/logfile) - gedit
logfile
logfile.txt

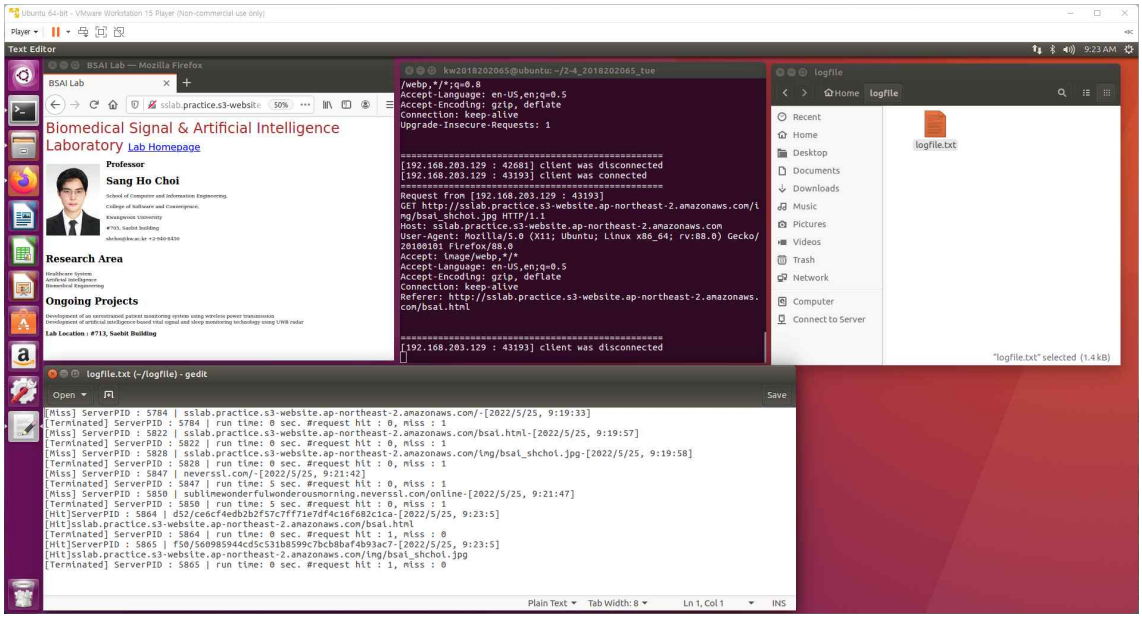
3.


Ubuntu 64-bit - VMware Workstation 15 Player (Non-commercial use only)
Player
Text Editor
9:20 AM
BSAI Lab — Mozilla Firefox
BSAI Lab
sslab.practice.s3-website
Biomedical Signal & Artificial Intelligence Laboratory
Lab Homepage
Professor
Sang Ho Choi
Research Area
Ongoing Projects
Lab Location : #713, Saebit Building
kw2018202065@ubuntu: ~/2-4_2018202065_tue
/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
[192.168.203.129 : 36025] client was disconnected
[192.168.203.129 : 37049] client was connected
Request from [192.168.203.129 : 37049]
GET http://sslab.practice.s3-website.ap-northeast-2.amazonaws.com/img/bsai_shchoi.jpg HTTP/1.1
Host: sslab.practice.s3-website.ap-northeast-2.amazonaws.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://sslab.practice.s3-website.ap-northeast-2.amazonaws.com/bsai.html
[192.168.203.129 : 37049] client was disconnected
logfile
Home
logfile
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Network
Computer
Connect to Server
logfile.txt
"logfile.txt" selected (565 bytes)
logfile.txt (~/logfile) - gedit
Open
Save
[Miss] ServerPID : 5784 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/-[2022/5/25, 9:19:33]
[Terminated] ServerPID : 5784 | run time: 0 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5822 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/bsai.html-[2022/5/25, 9:19:57]
[Terminated] ServerPID : 5822 | run time: 0 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5828 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/img/bsal_shchoi.jpg-[2022/5/25, 9:19:58]
[Terminated] ServerPID : 5828 | run time: 0 sec. #request hit : 0, miss : 1
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

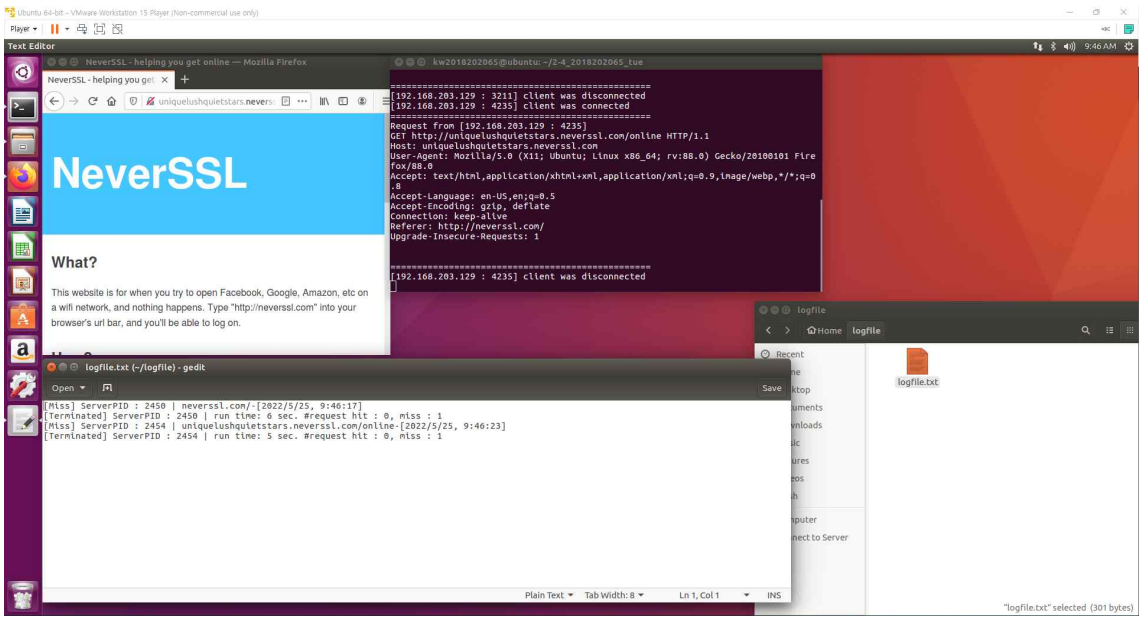
4.


Ubuntu 64-bit - VMware Workstation 15 Player (Non-commercial use only)
Player
Firefox Web Browser
9:22 AM
NeverSSL - helping you get online — Mozilla Firefox
NeverSSL - helping you get
sublimewonderfulwo
NeverSSL
What?
How?
Why?
kw2018202065@ubuntu: ~/2-4_2018202065_tue
Proxy-Connection: keep-alive
Connection: keep-alive
Host: push.services.mozilla.com:443
[192.168.203.129 : 41145] client was disconnected
[192.168.203.129 : 40121] client was disconnected
[192.168.203.129 : 41657] client was connected
Request from [192.168.203.129 : 41657]
GET http://sublimewonderfulwonderousmorning.neverssl.com/online HTTP/1.1
Host: sublimewonderfulwonderousmorning.neverssl.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://neverssl.com/
Upgrade-Insecure-Requests: 1
[192.168.203.129 : 41657] client was disconnected
logfile
Home
logfile
Recent
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Trash
Network
Computer
Connect to Server
logfile.txt
"logfile.txt" selected (878 bytes)
logfile.txt (~/logfile) - gedit
Open
Save
[Miss] ServerPID : 5784 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/-[2022/5/25, 9:19:33]
[Terminated] ServerPID : 5784 | run time: 0 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5822 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/bsai.html-[2022/5/25, 9:19:57]
[Terminated] ServerPID : 5822 | run time: 0 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5828 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/img/bsal_shchoi.jpg-[2022/5/25, 9:19:58]
[Terminated] ServerPID : 5828 | run time: 0 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5847 | neverssl.com/-[2022/5/25, 9:21:42]
[Terminated] ServerPID : 5847 | run time: 5 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5850 | sublimewonderfulwonderousmorning.neverssl.com/online-[2022/5/25, 9:21:47]
[Terminated] ServerPID : 5850 | run time: 5 sec. #request hit : 0, miss : 1
Plain Text
Tab Width: 8
Ln 1, Col 1
INS

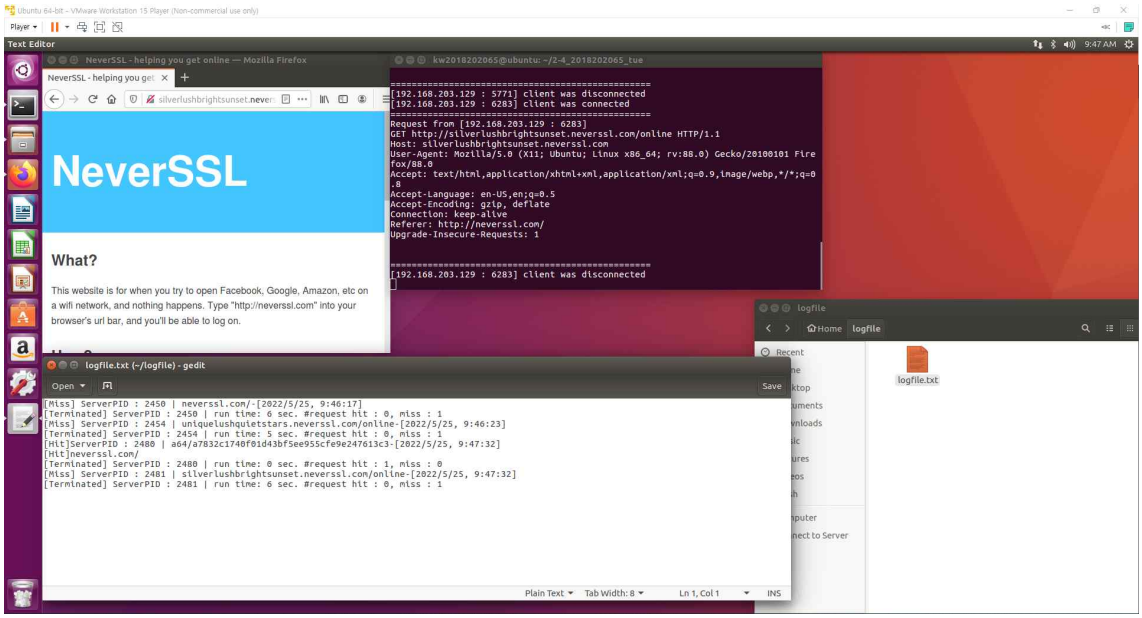
5.


Text Editor
BSAI Lab — Mozilla Firefox
Biomedical Signal & Artificial Intelligence Laboratory
Lab Homepage
Professor
Sang Ho Choi
Research Area
Ongoing Projects
kw2018202065@ubuntu: ~/2-4_2018202065_tue
/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
[192.168.203.129 : 42681] client was disconnected
[192.168.203.129 : 43193] client was connected
Request from [192.168.203.129 : 43193]
GET http://sslab.practice.s3-website.ap-northeast-2.amazonaws.com/img/bsai_shchoi.jpg HTTP/1.1
Host: sslab.practice.s3-website.ap-northeast-2.amazonaws.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://sslab.practice.s3-website.ap-northeast-2.amazonaws.com/bsai.html
[192.168.203.129 : 43193] client was disconnected
logfile
logfile.txt
logfile.txt (~/logfile) - gedit
[Miss] ServerPID : 5784 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/-[2022/5/25, 9:19:33]
[Terminated] ServerPID : 5784 | run time: 0 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5822 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/bsai.html-[2022/5/25, 9:19:57]
[Terminated] ServerPID : 5822 | run time: 0 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5828 | sslab.practice.s3-website.ap-northeast-2.amazonaws.com/img/bsai_shchoi.jpg-[2022/5/25, 9:19:58]
[Terminated] ServerPID : 5828 | run time: 0 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5847 | neverssl.com/-[2022/5/25, 9:21:42]
[Terminated] ServerPID : 5847 | run time: 5 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 5850 | sublimewonderfulwonderousmorning.neverssl.com/online-[2022/5/25, 9:21:47]
[Terminated] ServerPID : 5850 | run time: 5 sec. #request hit : 0, miss : 1
[Hit]ServerPID : 5864 | d52/ce6cf4edb2b2f57c7ff71e7df4c16f682c1ca-[2022/5/25, 9:23:5]
[Hit]sslab.practice.s3-website.ap-northeast-2.amazonaws.com/bsai.html
[Terminated] ServerPID : 5864 | run time: 0 sec. #request hit : 1, miss : 0
[Hit]ServerPID : 5865 | f50/56098594 4cd5c531b8599c7bcb8baf4b93ac7-[2022/5/25, 9:23:5]
[Hit]sslab.practice.s3-website.ap-northeast-2.amazonaws.com/img/bsai_shchoi.jpg
[Terminated] ServerPID : 5865 | run time: 0 sec. #request hit : 1, miss : 0

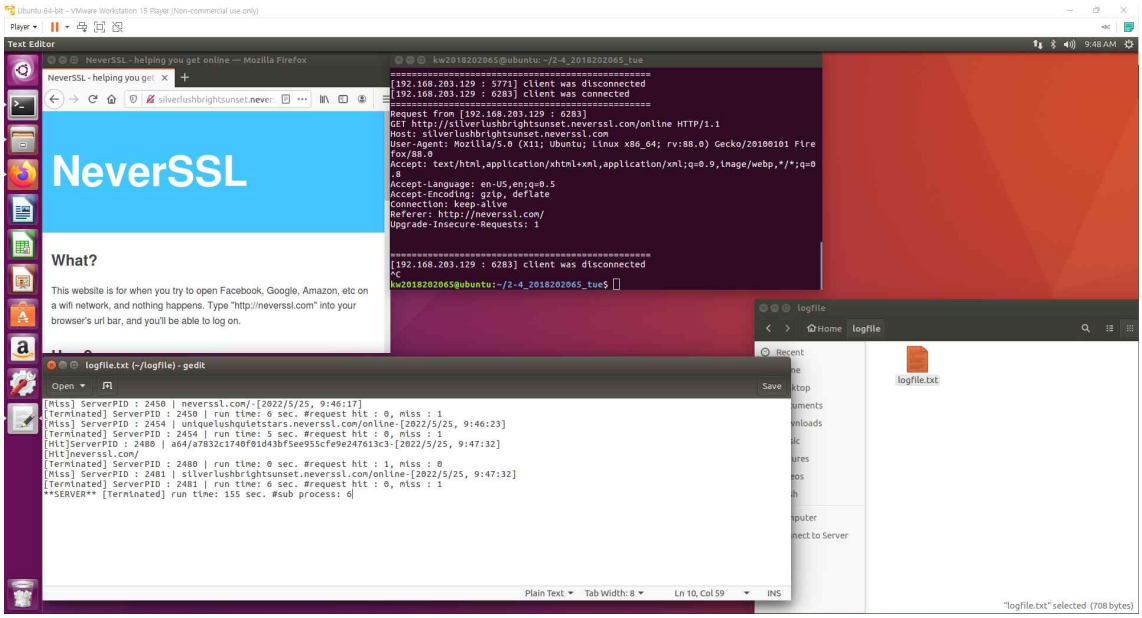
6.


Text Editor
NeverSSL - helping you get online — Mozilla Firefox
NeverSSL
What?
This website is for when you try to open Facebook, Google, Amazon, etc on a wifi network, and nothing happens. Type "http://neverssl.com" into your browser's url bar, and you'll be able to log on.
kw2018202065@ubuntu: ~/2-4_2018202065_tue
[192.168.203.129 : 3211] client was disconnected
[192.168.203.129 : 4235] client was connected
Request from [192.168.203.129 : 4235]
GET http://uniquelushquietstars.neverssl.com/online HTTP/1.1
Host: uniquelushquietstars.neverssl.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://neverssl.com/
Upgrade-Insecure-Requests: 1
[192.168.203.129 : 4235] client was disconnected
logfile
logfile.txt
logfile.txt (~/logfile) - gedit
[Miss] ServerPID : 2450 | neverssl.com/-[2022/5/25, 9:46:17]
[Terminated] ServerPID : 2450 | run time: 6 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 2454 | uniquelushquietstars.neverssl.com/online-[2022/5/25, 9:46:23]
[Terminated] ServerPID : 2454 | run time: 5 sec. #request hit : 0, miss : 1

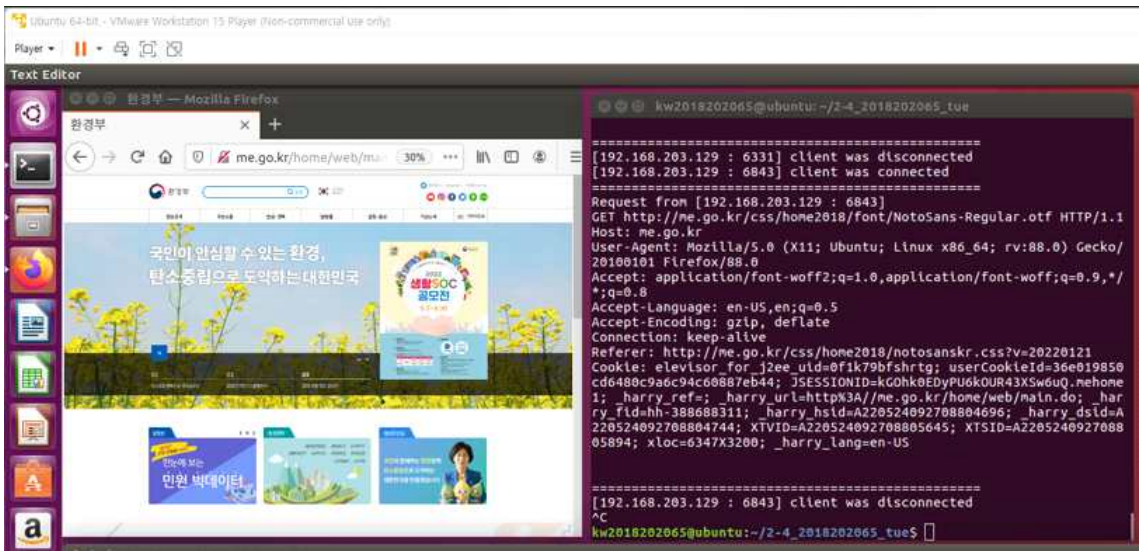
7.


NeverSSL
What?
This website is for when you try to open Facebook, Google, Amazon, etc on a wifi network, and nothing happens. Type "http://neverssl.com" into your browser's url bar, and you'll be able to log on.
[192.168.203.129 : 5771] client was disconnected
[192.168.203.129 : 6283] client was connected
Request from [192.168.203.129 : 6283]
GET http://silverlushbrightsunset.neverssl.com/online HTTP/1.1
Host: silverlushbrightsunset.neverssl.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://neverssl.com/
Upgrade-Insecure-Requests: 1
[192.168.203.129 : 6283] client was disconnected
logfile.txt (~/logfile) - gedit
[Miss] ServerPID : 2450 | neverssl.com/-[2022/5/25, 9:46:17]
[Terminated] ServerPID : 2450 | run time: 6 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 2454 | uniquelushquietstars.neverssl.com/online-[2022/5/25, 9:46:23]
[Terminated] ServerPID : 2454 | run time: 5 sec. #request hit : 0, miss : 1
[Hit]ServerPID : 2480 | a64/a7832c1740f01d43bf5ee955cfe9e247613c3-[2022/5/25, 9:47:32]
[Hit]neverssl.com/
[Terminated] ServerPID : 2480 | run time: 0 sec. #request hit : 1, miss : 0
[Miss] ServerPID : 2481 | silverlushbrightsunset.neverssl.com/online-[2022/5/25, 9:47:32]
[Terminated] ServerPID : 2481 | run time: 6 sec. #request hit : 0, miss : 1
logfile
logfile.txt

8.


NeverSSL
What?
This website is for when you try to open Facebook, Google, Amazon, etc on a wifi network, and nothing happens. Type "http://neverssl.com" into your browser's url bar, and you'll be able to log on.
[192.168.203.129 : 5771] client was disconnected
[192.168.203.129 : 6283] client was connected
Request from [192.168.203.129 : 6283]
GET http://silverlushbrightsunset.neverssl.com/online HTTP/1.1
Host: silverlushbrightsunset.neverssl.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://neverssl.com/
Upgrade-Insecure-Requests: 1
[192.168.203.129 : 6283] client was disconnected
^C
kw2018202065@ubuntu:~/2-4_2018202065_tue$
logfile.txt (~/logfile) - gedit
[Miss] ServerPID : 2450 | neverssl.com/-[2022/5/25, 9:46:17]
[Terminated] ServerPID : 2450 | run time: 6 sec. #request hit : 0, miss : 1
[Miss] ServerPID : 2454 | uniquelushquietstars.neverssl.com/online-[2022/5/25, 9:46:23]
[Terminated] ServerPID : 2454 | run time: 5 sec. #request hit : 0, miss : 1
[Hit]ServerPID : 2480 | a64/a7832c1740f01d43bf5ee955cfe9e247613c3-[2022/5/25, 9:47:32]
[Hit]neverssl.com/
[Terminated] ServerPID : 2480 | run time: 0 sec. #request hit : 1, miss : 0
[Miss] ServerPID : 2481 | silverlushbrightsunset.neverssl.com/online-[2022/5/25, 9:47:32]
[Terminated] ServerPID : 2481 | run time: 6 sec. #request hit : 0, miss : 1
SERVER [Terminated] run time: 155 sec. #sub process: 0
logfile
logfile.txt
"logfile.txt" selected (708 bytes)

9.


Ubuntu 64-bit - VMware Workstation 15 Player (Non-commercial use only)
Player
Text Editor
환경부 — Mozilla Firefox
환경부
me.go.kr/home/web/ma
30%
국민이 안심할 수 있는 환경,
탄소중립으로 도약하는 대한민국
생활SOC
공모전
민원 빅데이터
kw2018202065@ubuntu: ~/2-4_2018202065_tue
[192.168.203.129 : 6331] client was disconnected
[192.168.203.129 : 6843] client was connected
Request from [192.168.203.129 : 6843]
GET http://me.go.kr/css/home2018/font/NotoSans-Regular.otf HTTP/1.1
Host: me.go.kr
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
Accept: application/font-woff2;q=1.0,application/font-woff;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://me.go.kr/css/home2018/notosanskr.css?v=20220121
Cookie: elevisor_for_j2ee_uid=0f1k79bfshrtg; userCookieId=36e019850cd6480c9a6c94c60887eb44; JSESSIONID=kGOhk0EDyPU6kOUR43XSw6uQ.mehome1; _harry_ref=; _harry_url=http%3A//me.go.kr/home/web/main.do; _harry_fid=hh-388688311; _harry_hsid=A220524092708804696; _harry_dsid=A220524092708804744; XTVID=A220524092708805645; XTSID=A220524092708805894; xloc=6347X3200; _harry_lang=en-US
[192.168.203.129 : 6843] client was disconnected
^C
kw2018202065@ubuntu:~/2-4_2018202065_tue$

● 고찰

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

1. 웹 서버와 socket통신 과정에서 read 함수를 사용하였을 시 다 읽었어서 0을 반환하기 전 대기하는 상태

웹 서버의 response 메시지를 읽기 위해 read 함수를 사용하였다. 하지만 response 메시지를 다 읽었음에도 불구하고 read의 리턴값이 0이 되는 시간이 오래 지속되는 것을 발견하였다. 이에 대한 이유를 알아보니 웹 서버와 클라이언트 간의 소켓 통신에 있어서 클라언트가 웹 서버의 response를 다 읽었다 하더라도 read는 사이트 마다 지정된 시간동안 서버에 read를 대기 할 것이고 이를 예방하기 위해선 클라이언트 측에서 웹서버에 대한 소켓을 종료하여야 read가 0을 반환한다는 것이다. 그래서 이를 위해 먼저 프로그램 구현 가장 초기에 while ((read_num= read(server_fd, buf, BUFFSIZE))>0)으로 response를 받는 방식을 사용하였고 서버측으로부터 완전한 response를 받았을 경우를 생각해 보았다. 이때 얻은 결론은 보통의 response 메시지에 있어서는 content-length 값을 헤더 라인에 포함하게 될 것이고 이를 추출하여 컨텐츠 길이를 알아낸 다음 read의 반환값을 축척하여 컨텐츠 길이만큼 추척되었을 경우 다 읽었다고 가정하였다. 하지만 이는 오류가 있다. 컨텐츠 길이에는 헤더라인의 길이는 포함되어 있지 않기 때문에 이와 같은 방법을 사용하기 위해서는 헤더라인의 길이도 알아내어야 한다. 이를 해결하기 위해 다음과 같은 방식의 코드를 작성하였다.

먼저 헤더라인을 buf변수로 읽고 그 부분을 strtok함수를 통해 "content-length:" 까지 끊고 그 뒤에 나온 컨텐츠 길이 값을 추출하고 따로 저장한 다음 entity-body에서 content가 읽기 전 entity-body의 끝 까지의 길이를 추출하여 준다. 이를 코딩으로 표현하면 다음과 같다.

```
if (check_num == 1)//check_num이 1인 경우 response메시지의 가장 첫 번째 부분 즉 헤더 부분임으로 여기에 컨텐츠-length가 있다.
{
        char tmp1[BUFFSIZE] = { 0, };
        char tmp2[BUFFSIZE] = { 0, };
        char* tok1 = NULL;
        char* tok2 = NULL;
        strcpy(tmp1, buf);
        strcpy(tmp2, buf);
        tok1 = strtok(tmp1, "₩n");
        Content_Length_before_entity_body += strlen(tok1);
        newline_count++;
        //newline_count++;의 뜻은 개행을 기준으로 끝고 그 길이를 리턴 받는데
        //개행이 포함되지 않아 최종적인 길이에는 개행의 개수가 포함되지 않는다. 고로
        //개행의 개수를 카운트하여 길이에 추가하여 준다.
        while (strcmp(tok1, "₩r") != 0)
        {
                //헤더부분의 길이를 strtok로 알아낸다.
                tok1 = strtok(NULL, "₩n");
                Content_Length_before_entity_body += strlen(tok1);
                newline_count++;
        }
        tok2 = strtok(tmp2, " ");
```

```
while(strcmp(tok2, "Content-Length:") != 0)// Content-Length: 글자 탐색
{


        if (strcmp(tok2, "Transfer-Encoding:") == 0)//Transfer-Encoding: 글자 탐색
        //Trans-encoding:의 글자가 발견되면 content-length를 포함하지 않아 break해준다.
        //이부분이 없으면 무한 루프에 빠진다.
        {
                chunck_check = true;
                break;
        }
        tok2 = strtok(NULL, "₩n");
        tok2 = strtok(NULL, " ");
}
tok2 = strtok(NULL, "₩r");
char* Content_Length_char = NULL;
Content_Length_char = tok2;
Content_Length_int = atoi(Content_Length_char);
//컨텐츠 길이의 숫자가 알맞게 추출되었는지 확인을 위한 출력문
//printf("Content_Length_int is %d ₩n", Content_Length_int);
//entity body 전까지의 리스폰스 메세지의 길이를 알맞게 계산했는 지 확인을 위한 출력문
//printf("Content_Length_before_entity_body is %d ₩n", Content_Length_before_entity_body);
//결과적으로 컨텐츠-length 포함 response 메시지의 길이는 다음과 같다.
// response 메시지 길이 = 컨텐츠 길이 + 헤더까지 끊으면서 축척한 길이 + 헤더 까지 끊으면서 만난
//개행의 개수

}
```

이렇게 추출한 값을 아래의 코딩을 통해

while ((read_num= read(server_fd, buf, BUFFSIZE))>0)을 탈출하였다.

```
if ((Content_Length_int + Content_Length_before_entity_body+ newline_count <= read_num_sum)&&(chunck_check==false))
// response 메시지 길이 = 컨텐츠 길이 + 헤더까지 끊으면서 축척한 길이 + 헤더 까지 끊으면서 만난
//개행의 개수
{
        close(server_fd);
}
```

2. transfer-encoding: chunk일 때 read함수의 대기상태 해결 문제

transfer-encoding: chunk일 때는 청크단위의 데이터가 묶음이 크기를 지정하지 않고 오기 때문에 컨텐츠 길이 항목이 따로 없어 이를 해결하기 위해 위에서 구현한 코드의 read함수의 사용에 있어서 timeout을 걸어두기로 하였다. seleck함수와 소켓에 timeout을 추가하는 위에서 설명한 read_with_timeout() 을 사용하여 해결하였다. 즉 기존의

while ((read_num= read(server_fd, buf, BUFFSIZE))>0) 을

while ((read_num= read_with_timeout(server_fd, buf, BUFFSIZE,5000))>0) 로 변경하여 chunk데이터로 오는 response에서 다 받아서 0을 반환하기까지 대기 시간이 지속될 때 5초 이상 걸릴 시 read에서 0을 반환하여 종료하는 방식으로 코딩하였다 하지만 이것은 완벽한 방식이 아니다. 만약 사이트 내의 어떠한 부분이 read_with_timeout 함수에 설정된 시간보다 길게 소요되는 부분이 있다면 read를 리턴해버리는 문제가 발생하여 이것을 해결하기 위해선 타임 아웃의 시간을 적절하게 가져가야 하는데 이는 상대적인 것이라 이번 과제를 진행

하면서 제공된 예시의 사이트에서는 5초의 타임아웃이면 청크 데이터 예시 사이트 (neverssl.com)을 읽을 수 있기에 5초로 설정하였다.

3. SIGINT(Ctrl+C)가 들어왔을 때 만약 서브 프로세스가 진행 중인 것이 있을 때
먼저 다중 이용 방식의 캐시에 있어서 서브 프로세스가 진행 중일 때 프로그램이 종료가 되는 경우가 무조건 발생하기 때문에 이러한 부분을 처리해주어야 하고 다음과 같은 방버을 통해 해결하였다. 먼저 전역변수로 부모 프로세스 아이디 즉 프로그램 프로세스 아이디를 저장할 변수를 선언한다. 이후 main함수가 실행되면 부모 프로세스 아이디 즉 프로그램 프로세스 아이디가 생성되고 이를 geptid() 함수를 통해 선언한 전역변수에 저장하고 프로그램 동작 중 SIGINT(Ctrl+C) 신호를 입력 받게 되면 시그널 핸들러 함수로 가고 SIGINT시그널에 대한 동작을 할 것이다. 이때 SIGINT 시그널은 자식 프로세스와 부모 프로세스 둘다 실행할 것이고 핸들러에서 전역변수에서 저장한 부모 프로세스 아이디와 핸들러 함수 안에서 얻은 프로세스 아이디를 비교한다. 이후 이 값이 같다면 로그파일에 종료문을 입력하고 exit(0)하고 이값과 다르다면 바로 exit(0) 한다. 이를 테스트 해보니 보통 실행 중인 자식 프로세스 보다 부모 프로세스의 종료가 더 빨랐고 이후 자식 프로세스는 종료하지만 부모가 종료됨으로 부모 프로세스에서 실행하는 SIGCHLD를 실행하지 못하고 또한 고아 프로세스로 남다 init프로세스에 의해 종료값이 반환됨을 알 수 있었다.