

네트워크 프로그래밍(가)

1차 과제

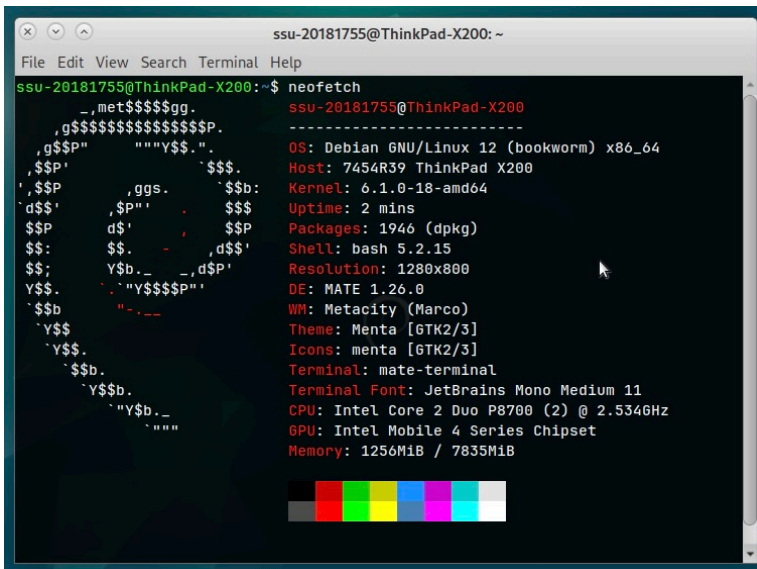
일어일문학과

20181755 이건희

2024년 04월

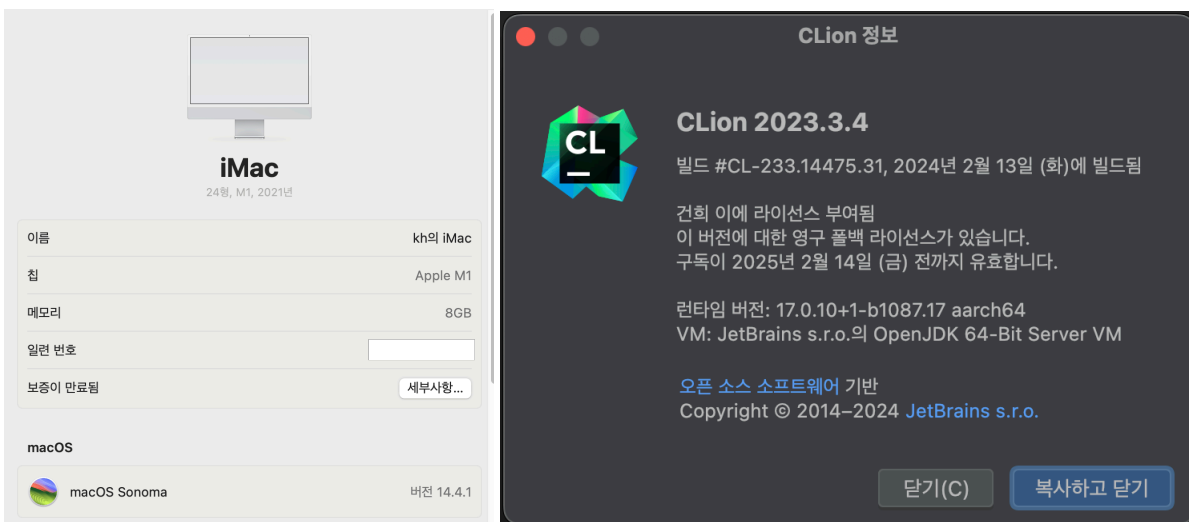
1. 환경

1-1. 실행



Debian 12 x86_64 @ ThinkPad X200

1-2. 편집



macOS Sonoma @ iMac M1

CLion 2023.3

1-3. 기타

공부한것을 보고서에 정리하여 나중에 다시 읽어보는 습관이 있어, 과제 치고는 불필요하게 길니다. 미리 양해를 부탁드립니다. 실행결과는 가장 마지막 페이지에 있습니다.

2. 프로그램 설명

코드의 순서대로 설명하려면 부자연스럽기에, 실행 흐름에 따라 설명하겠다.

2-1. headers, preprocessor, typedef

```
hw1_20181755.c
1  #include <sys/socket.h>
2
3  #include <netinet/in.h>
4  #include <arpa/inet.h>
5  #include <netdb.h>
6
7  #include <stdio.h>
8  #include <string.h>
9
10 typedef struct addrinfo adin;
```

필요한 헤더 파일들을 포함시키고, struct addrinfo를 매 번 치는것이 귀찮아 adin으로 typedef를 해주었다. 이후의 설명에서 처음 등장하는 내용들은 어느 헤더파일에서 가져온것인지 부연설명하겠다. 사용하는 CLion에서 Command + B면 바로 열어볼 수 있다.

- struct addrinfo : netdb.h

```
netdb.h
이 파일은 프로젝트 타겟에 포함되지 않으므로 코드 분석 기능이 제대로 작동하지 않을 수 있습니다.
146
147 struct addrinfo {
148     int ai_flags; /* AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST */
149     int ai_family; /* PF_xxx */
150     int ai_socktype; /* SOCK_xxx */
151     int ai_protocol; /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
152     socklen_t ai_addrlen; /* length of ai_addr */
153     char *ai_canonname; /* canonical name for hostname */
154     struct sockaddr *ai_addr; /* binary address */
155     struct addrinfo *ai_next; /* next structure in linked list */
156 };
157
```

주소정보를 저장하는 구조체이다. 마지막 멤버는 다음 struct addrinfo를 가리키기에, linked list 자료구조로 되어 있다. 노드의 멤버들을 알아보겠다.

- ai_flags : netdb.h

```
#define AI_PASSIVE 0x00000001 /* get address to use bind() */
#define AI_CANONNAME 0x00000002 /* fill ai_canonname */
#define AI_NUMERICHOST 0x00000004 /* prevent host name resolution */
```

코드에서는 수동적으로 듣기 위해 AI_PASSIVE가 나온다. 들어갈 수 있는것들을 비트로 바꿔보면 0001, 0010, 0100인데, 운영체제처럼 flag를 비트단위로 설정하여 내부적으로 처리한다 생각할 수 있다.

- ai_family : socket.h

```
#define AF_UNSPEC      0          /* unspecified */
#define AF_UNIX        1          /* local to host (pipes) */
#if !defined(_POSIX_C_SOURCE) || defined(_DARWIN_C_SOURCE)
#define AF_LOCAL        AF_UNIX   /* backward compatibility */
#endif /* (!_POSIX_C_SOURCE || _DARWIN_C_SOURCE) */
#define AF_INET        2          /* internet: UDP, TCP, etc. */
#if !defined(_POSIX_C_SOURCE) || defined(_DARWIN_C_SOURCE)
#define AF_IMPLINK      3          /* arpanet imp addresses */
#define AF_PUP          4          /* pup protocols: e.g. BSP */
#define AF_CHAOS        5          /* mit CHAOS protocols */
#define AF_NS           6          /* XEROX NS protocols */
#define AF_ISO          7          /* ISO protocols */
#define AF_OSI          AF_ISO
#define AF_ECMA         8          /* European computer manufacturers */
#define AF_DATAKIT      9          /* datakit protocols */
#define AF_CCITT        10         /* CCITT protocols, X.25 etc */
#define AF_SNA          11         /* IBM SNA */
#define AF_DECnet       12         /* DECnet */
#define AF_DLI          13         /* DEC Direct data link interface */
#define AF_LAT          14         /* LAT */
#define AF_HYLINK      15         /* NSC Hyperchannel */
#define AF_APPLETALK    16         /* Apple Talk */
#define AF_ROUTE        17         /* Internal Routing Protocol */
#define AF_LINK         18         /* Link layer interface */
#define pseudo_AF_XTP   19         /* eXpress Transfer Protocol (no AF) */
#define AF_COIP         20         /* connection-oriented IP, aka ST II */
#define AF_CNT          21         /* Computer Network Technology */
#define pseudo_AF_RTIP  22         /* Help Identify RTIP packets */
#define AF_IPX          23         /* Novell Internet Protocol */
#define AF_SIP          24         /* Simple Internet Protocol */
#define pseudo_AF_PIP   25         /* Help Identify PIP packets */
#define AF_NDRV         27         /* Network Driver 'raw' access */
#define AF_ISDN         28         /* Integrated Services Digital Network */
#define AF_E164         AF_ISDN   /* CCITT E.164 recommendation */
#define pseudo_AF_KEY   29         /* Internal key-management function */
#endif /* (!_POSIX_C_SOURCE || _DARWIN_C_SOURCE) */
#define AF_INET6        30         /* IPv6 */
#if !defined(_POSIX_C_SOURCE) || defined(_DARWIN_C_SOURCE)
#define AF_NATM         31         /* native ATM access */
#define AF_SYSTEM       32         /* Kernel event messages */
#define AF_NETBIOS      33         /* NetBIOS */
#define AF_PPP          34         /* PPP communication protocol */
#define pseudo_AF_HDRCMPLT 35      /* Used by BPF to not rewrite headers
                                     * in interface output routine */
#define AF_RESERVED_36  36         /* Reserved for internal usage */
#define AF_IEEE80211    37         /* IEEE 802.11 protocol */
#define AF_UTUN         38         /* User Datagram Protocol */
#define AF_VSOCK        40         /* VM Sockets */
#define AF_MAX          41
```

Address Family를 의미한다. 코드에서는 IPv4, IPv6를 쿼리할것이기때 AF_INET, AF_INET6이 나온다. 이 외에도 다양한 Address Family들이 존재한다.

- ai_socktype : socket.h

```
#define SOCK_STREAM    1          /* stream socket */
#define SOCK_DGRAM     2          /* datagram socket */
#define SOCK_RAW       3          /* raw-protocol interface */
```

소켓 종류를 의미한다. 1은 연결지향적인 stream socket, 2는 비연결지향적인 datagram socket, 3은 raw protocol한 소켓이다.

- ai_protocol : in.h

```
#define IPPROTO_IP 0 /* dummy for IP */
#if !defined(_POSIX_C_SOURCE) || defined(_DARWIN_C_SOURCE)
#define IPPROTO_HOPOPTS 0 /* IP6 hop-by-hop options */
#endif /* (!_POSIX_C_SOURCE || _DARWIN_C_SOURCE) */
#define IPPROTO_ICMP 1 /* control message protocol */
#if !defined(_POSIX_C_SOURCE) || defined(_DARWIN_C_SOURCE)
#define IPPROTO_IGMP 2 /* group mgmt protocol */
#define IPPROTO_GGP 3 /* gateway^2 (deprecated) */
#define IPPROTO_IPV4 4 /* IPv4 encapsulation */
#define IPPROTO_IPIP IPPROTO_IPV4 /* for compatibility */
#endif /* (!_POSIX_C_SOURCE || _DARWIN_C_SOURCE) */
#define IPPROTO_TCP 6 /* tcp */
#if !defined(_POSIX_C_SOURCE) || defined(_DARWIN_C_SOURCE)
#define IPPROTO_ST 7 /* Stream protocol II */
#define IPPROTO_EGP 8 /* exterior gateway protocol */
#define IPPROTO_PIGP 9 /* private interior gateway */
#define IPPROTO_RCCMON 10 /* BBN RCC Monitoring */
#define IPPROTO_NVP II 11 /* network voice protocol*/
#define IPPROTO_PUP 12 /* pup */
#define IPPROTO_ARGUS 13 /* Argus */
#define IPPROTO_EMCON 14 /* EMCON */
#define IPPROTO_XNET 15 /* Cross Net Debugger */
#define IPPROTO_CHAOS 16 /* Chaos*/
#endif /* (!_POSIX_C_SOURCE || _DARWIN_C_SOURCE) */
#define IPPROTO_UDP 17 /* user datagram protocol */
```

(너무 길어서 내용 일부 생략)

프로토콜 종류를 의미한다. TCP의 경우 IPPROTO_TP, UDP의 경우 IPPROTO_UDP를 사용한다. 그 외에도 ICMP, IGMP같은 어디서 들어본 프로토콜들도 있으니, 특정 프로토콜을 원할 경우 참고하여 바꾸면 된다.

- ai_addrln :

```
#ifndef _SOCKLEN_T
#define _SOCKLEN_T

#include <machine/types.h> /* __darwin_socklen_t */
typedef __darwin_socklen_t socklen_t;
#endif
```

_socklen.h

```
typedef unsigned long __darwin_clock_t;
typedef __uint32_t __darwin_socklen_t;
typedef long __darwin_ssize_t;
typedef long __darwin_time_t;
```

```
typedef unsigned char __uint8_t;
typedef short __int16_t;
typedef unsigned short __uint16_t;
typedef int __int32_t;
typedef unsigned int __uint32_t;
typedef long long __int64_t;
typedef unsigned long long __uint64_t;
```

_types.h

결국 ai_addrln의 자료형인 __socklen_t는 unsigned int이다.

혹자는 이 내용이 무의미하다 할수도 있겠으나, 자료형을 확실히 정해두는것은 프로그래밍의 기본중 기본이다. 해당 내용은 system-specific할 수 있다.

- ai_canonname : CNAME

- ai_next : 다음 구조체를 가리키는 linked list 노드의 멤버

2-2. argc 검사, hints 정의

```
70 ▶ int main(int argc, char *argv[]) {
71     if(argc < 2) {
72         fprintf(stderr, "usage: getaddrinfo hostname\n");
73         return 1;
74     }
75
76     char hostname[100];
77     strcpy(hostname, argv[1]); // argv[1](터미널에서 입력값)에서 검색 원하는 주소 복사
78
79     adin hints4;
80     adin hints6;
81
82     memset(&hints4, 0, sizeof hints4);
83     memset(&hints6, 0, sizeof hints6);
84
85     hints4.ai_flags = AI_PASSIVE; // 모든 인터페이스에서 들어오는것을 다 듣겠다.
86     hints4.ai_family = AF_INET; // IPv4만 받겠다.
87     hints4.ai_socktype = SOCK_STREAM; // TCP
88
89     hints6.ai_flags = AI_PASSIVE; // 모든 인터페이스에서 들어오는것을 다 듣겠다.
90     hints6.ai_family = AF_INET6; // IPv6만 받겠다.
91     hints6.ai_socktype = SOCK_STREAM; // TCP
92 }
```

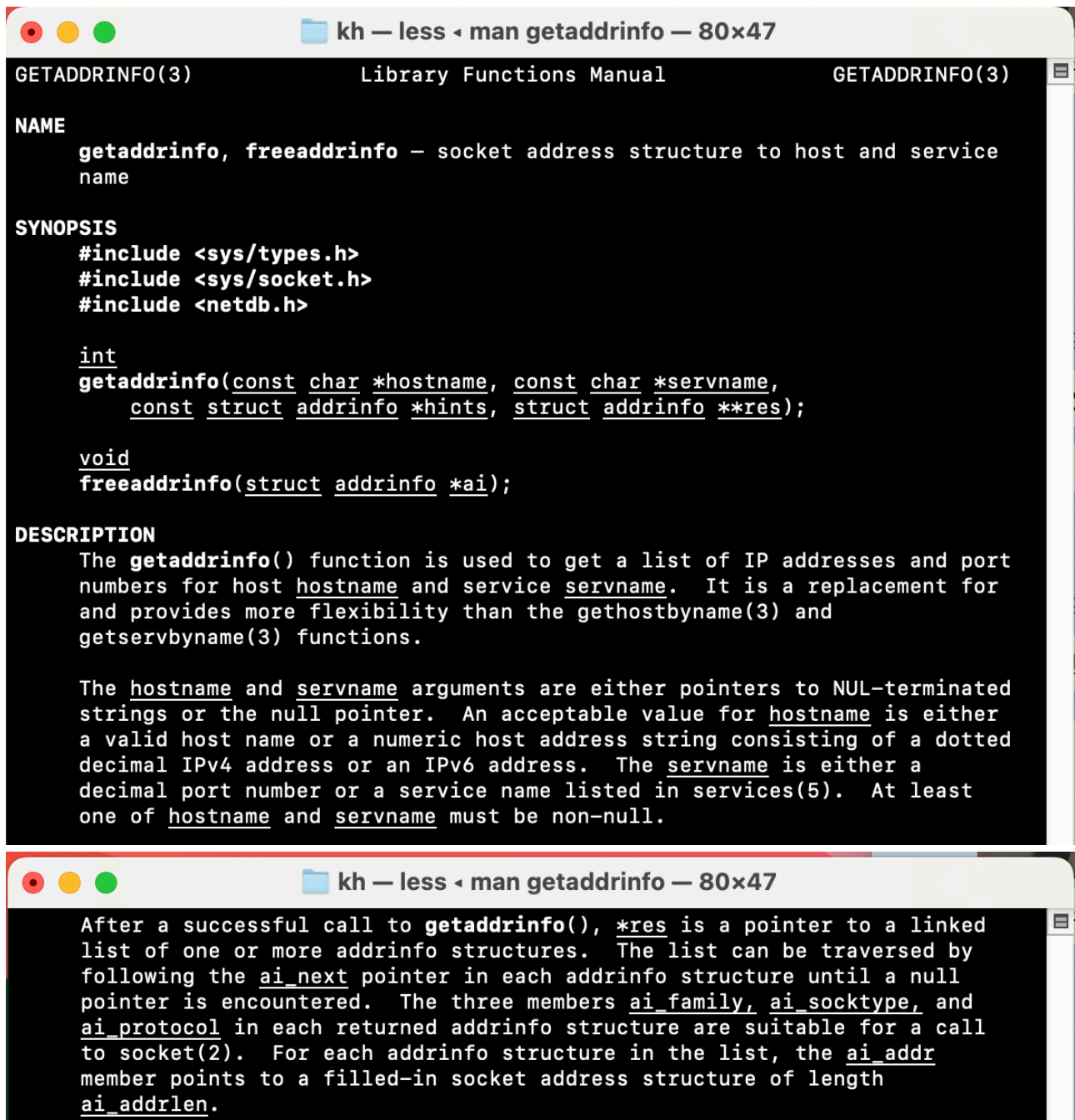
argc가 2 미만일 경우 argc[1], 즉 질의하려는 사이트를 사용자가 넣지 않았다는 뜻이므로 경고를 띄우고 프로그램을 종료시킨다.

사실 굉장히 단순로운 과제라서, 연습할겸 v4, v6의 경우를 분리하여 hints를 정의했다. hints4는 IPv4, hints6은 IPv6을 대상으로 하는 adin(struct addrinfo)이다. 이어서 memset으로 해당 메모리 영역을 초기화 시킨다. 해당 구조체들에 2-1에서 설명한 값들중 조건에 맞도록 멤버를 설정한다.

2-3. getaddrinfo

```
96      // 주소 정보 불러옴, 에러처리: 0이 아닐경우 문제 있음
97      int status4 = getaddrinfo(hostname, NULL, &hints4, &res4); // IPv4
98      int status6 = getaddrinfo(hostname, NULL, &hints6, &res6); // IPv6
99
100     int errsel = errorHandler(status4, status6);
```

이를 이해하기 위해서는 getaddrinfo 함수에 대해 이해할 필요가 있다.



```
GETADDRINFO(3)                                Library Functions Manual                                GETADDRINFO(3)

NAME
    getaddrinfo, freeaddrinfo - socket address structure to host and service
    name

SYNOPSIS
    #include <sys/types.h>
    #include <sys/socket.h>
    #include <netdb.h>

    int
    getaddrinfo(const char *hostname, const char *servname,
                const struct addrinfo *hints, struct addrinfo **res);

    void
    freeaddrinfo(struct addrinfo *ai);

DESCRIPTION
    The getaddrinfo() function is used to get a list of IP addresses and port
    numbers for host hostname and service servname. It is a replacement for
    and provides more flexibility than the gethostbyname(3) and
    getservbyname(3) functions.

    The hostname and servname arguments are either pointers to NUL-terminated
    strings or the null pointer. An acceptable value for hostname is either
    a valid host name or a numeric host address string consisting of a dotted
    decimal IPv4 address or an IPv6 address. The servname is either a
    decimal port number or a service name listed in services(5). At least
    one of hostname and servname must be non-null.

    After a successful call to getaddrinfo(), *res is a pointer to a linked
    list of one or more addrinfo structures. The list can be traversed by
    following the ai_next pointer in each addrinfo structure until a null
    pointer is encountered. The three members ai_family, ai_socktype, and
    ai_protocol in each returned addrinfo structure are suitable for a call
    to socket(2). For each addrinfo structure in the list, the ai_addr
    member points to a filled-in socket address structure of length
    ai_addrlen.
```




host name(ex. www.google.com), service name(ex. http), hints, res를 받아서, hostname에 해당하는 주소를 linked list인 res에 저장한다. 성공시 0을 return, 실패시 에러코드에 해당하는 값을 반환한다.

status4, status6은 각각 getaddrinfo에서 반환받은 IPv4의 상태, IPv6의 상태이다.
이들을 errorHandler 함수에 매개변수로 집어넣는다.

```
12 int errorHandler(int status4, int status6) {
13     int errCase = 0;
14
15     int ret = -1;
16     if(status4 == 0 && status6 == 0) {        // 문제 없으면
17         ret = 0;
18     }
19     else if(status4 != 0 && status6 != 0) { // 둘 다 문제있으면
20         fprintf(stderr, "getaddrinfo IPv4: %s\n", gai_strerror(status4));
21         fprintf(stderr, "getaddrinfo IPv6: %s\n", gai_strerror(status6));
22         ret = 3;
23     }
24     else {
25         if(status4 == 0) {    // v6만 오류시
26             fprintf(stderr, "getaddrinfo IPv6: %s\n", gai_strerror(status6));
27             ret = 1;
28         }
29         else {                // v4만 오류시
30             fprintf(stderr, "getaddrinfo IPv4: %s\n", gai_strerror(status4));
31             ret = 2;
32         }
33     }
34
35     return ret;
36 }
```

status4/status6, 0임(에러 없음)/0이 아님(에러 발생)으로 나올 수 있는 경우의 수는 4가지이다.

우선 각 경우의 수에 맞는 에러에 대한 설명을 출력한다.

status4 == 0 && status6 == 0 : 출력할 오류 없음

status4 != 0 && status6 != 0 : IPv4, IPv6에 대한 오류 출력

status4 == 0 && status6 != 0 : IPv6에 대한 오류 출력

status4 != 0 && status6 == 0 : IPv4에 대한 오류 출력

다음으로 각 경우가 어떤 상황인지에 대한 값을 반환한다. 반환값의 의미는 아래와 같다.

0: 문제 없음

1: IPv4 문제없음, IPv6 문제

2: IPv4 문제, IPv6 문제없음

3: 둘 다 문제있음

errHandler가 반환한 값은 errsel에 저장되고, errsel은 switch문에 들어간다.

```
102     switch(errsel) {
103     case 0: { // 둘 다 오류없음!
104         prnInet(res4);
105         prnInet6(res6);
106         freeaddrinfo(res4);
107         freeaddrinfo(res6);
108         break;
109     }
110     case 1: { // v6만 오류, v4 정보 출력
111         prnInet(res4);
112         freeaddrinfo(res4);
113         break;
114     }
115     case 2: { // v4만 오류, v6 정보 출력
116         prnInet6(res6);
117         freeaddrinfo(res6);
118         break;
119     }
120     default;;
121 }
122
123 return 0;
124 }
```

case 0: 둘 다 오류가 없을경우

IPv4 주소들, IPv6 주소들을 출력하고, res4, res6에 대해 freeaddrinfo를 한다

case 1: IPv4 문제없고, IPv6가 문제인 경우

IPv4 주소들을 출력하고, res4에 대해 freeaddrinfo를 한다

case 2: IPv6 문제없고, IPv4가 문제인 경우

IPv6 주소들을 출력하고, res6에 대해 freeaddrinfo를 한다.

그외: 둘 다 문제인 경우

아무것도 안한다

freeaddrinfo 함수에 대해 주의해야 할 것 한가지를 짚고가겠다.

getaddrinfo 함수는 작동 과정에서 내부적으로 각 노드를 동적할당한다. freeaddrinfo는 getaddrinfo에서 동적할당 해준 연결리스트의 노드들을 따라가면서 해제해주는 함수다.

freeaddrinfo의 매개변수는 struct addrinfo *형이다. 코드에서는 이 자리에 res4와 res6을 집어넣는데, 위에서 말했듯 이는 linked list의 노드이다.

코드의 마지막에

```
freeaddrinfo(res4);
```

```
freeaddrinfo(res6);
```

와 같이 일괄적으로 freeaddrinfo를 하겠다 하자.

argv[1]로 www.naver.com(마지막장의 결과에 있는 내용이지만, 네이버는 아직 IPv6주소가 없다)을 넣었을경우, status6은 2이고, res6은 아무것도 할당받지 못한 채 쓰레기값을 담고있을것이다.

이 상황에서 freeaddrinfo(res6)을 할 경우, 할당하지 않은 곳을 해제하려는것과 똑같고, 이는 인터럽트 시그널을 발생시킨다.

할당하지 않은곳을 해제하면 안된다. 내가 했던 실수다.

IPv4와 IPv6의 출력 함수를 따로 정의해주었다.

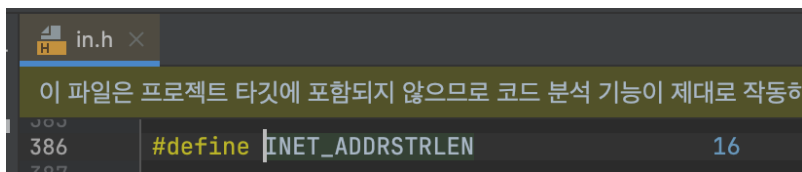
```
38 void prnInet(adin *res4) { // IPv4
39     char ipstr[INET_ADDRSTRLEN]; // INET_ADDRSTRLEN = 16 (in.h)
40     adin *p4;
41
42     // linkedlist 정보 탐색
43     for(p4 = res4; p4 != NULL; p4 = p4->ai_next) {
44         void *addr;
45
46         struct sockaddr_in *ipv4 = (struct sockaddr_in *) p4->ai_addr;
47         addr = &(ipv4->sin_addr);
48
49         inet_ntop(p4->ai_family, addr, ipstr, sizeof ipstr);
50         printf("- IPv4: %s\n", ipstr);
51     }
52 }
53
54 void prnInet6(adin *res6) { // IPv6
55     char ipstr[INET6_ADDRSTRLEN]; // INET6_ADDRSTRLEN = 46 (in6.h)
56     adin *p6;
57
58     // linkedlist 정보 탐색
59     for(p6 = res6; p6 != NULL; p6 = p6->ai_next) {
60         void *addr;
61
62         struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *) p6->ai_addr;
63         addr = &(ipv6->sin6_addr);
64
65         inet_ntop(p6->ai_family, addr, ipstr, sizeof ipstr);
66         printf("- IPv6: %s\n", ipstr);
67     }
68 }
```

prnInet, prnInet6은 각각 IPv4, IPv6을 출력하는 함수이다.

작동원리는 수업시간에 배웠기에 따로 설명하지는 않겠다. 다만 알고가야할 것들이 몇 가지 있다.

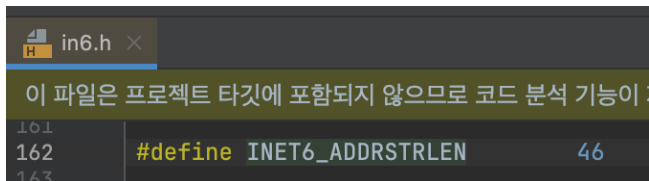
```
374 struct sockaddr_in {
375     __uint8_t sin_len;
376     sa_family_t sin_family;
377     in_port_t sin_port;
378     struct in_addr sin_addr;
379     char sin_zero[8];
380 };
```

struct sockaddr_in은 in.h에 정의되어있다.



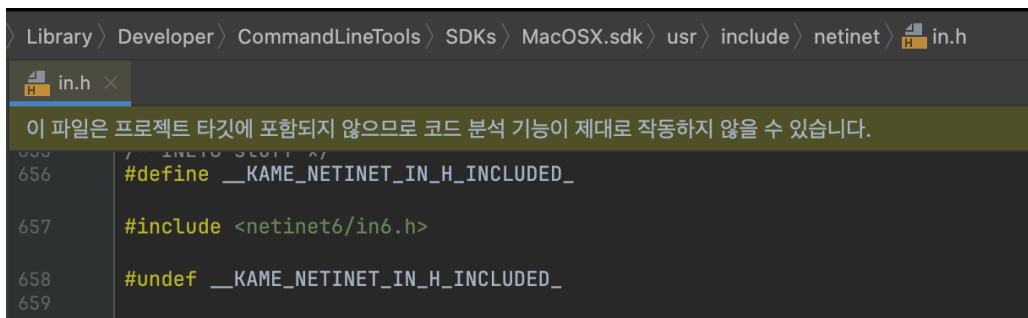
```
in.h x
이 파일은 프로젝트 타겟에 포함되지 않으므로 코드 분석 기능이 제대로 작동하
386 #define INET_ADDRSTRLEN 16
387
```

INET_ADDRSTRLEN은 16으로 in.h에 전처리되어있다.



```
in6.h x
이 파일은 프로젝트 타겟에 포함되지 않으므로 코드 분석 기능이 제대로 작동하
161
162 #define INET6_ADDRSTRLEN 46
163
```

INET6_ADDRSTRLEN은 46으로 in6.h에 전처리되어있다.



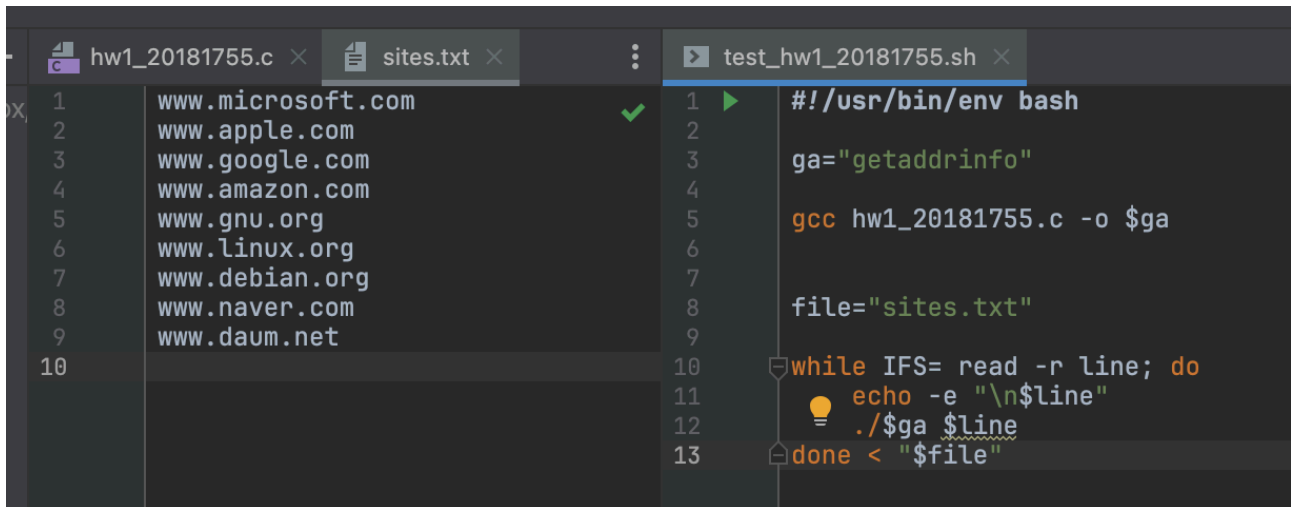
```
Library > Developer > CommandLineTools > SDKs > MacOSX.sdk > usr > include > netinet > in.h
in.h x
이 파일은 프로젝트 타겟에 포함되지 않으므로 코드 분석 기능이 제대로 작동하지 않을 수 있습니다.
656 #define __KAME_NETINET_IN_H_INCLUDED_
657 #include <netinet6/in6.h>
658 #undef __KAME_NETINET_IN_H_INCLUDED_
659
660
```

흥미로운것은, in.h에 in6.h도 포함하게끔 지시문이 있는점이다.

모든 작업이 완료되었으면 정상적으로 프로그램을 종료한다.

3. 작동 확인

3-1. 들어가기 전에



```
hw1_20181755.c  sites.txt  test_hw1_20181755.sh
1 www.microsoft.com
2 www.apple.com
3 www.google.com
4 www.amazon.com
5 www.gnu.org
6 www.linux.org
7 www.debian.org
8 www.naver.com
9 www.daum.net
10
1 #!/usr/bin/env bash
2
3 ga="getaddrinfo"
4
5 gcc hw1_20181755.c -o $ga
6
7 file="sites.txt"
8
9 while IFS= read -r line; do
10     echo -e "\n$line"
11     ./$ga $line
12 done < "$file"
```

주소를 하나하나 입력하는것이 귀찮아서, shell script를 하나 만들었다.

과제 폴더에서

```
sudo chmod 766 test_hw1_20181755.sh
```

```
./test_hw1_20181755.sh
```

하면 알아서 실행이 될 것이다.

과제폴더에서 sites.txt를 바꾸면, 다른 그 어떤 주소로도 테스트가 가능하다.

site.txt를 스크립트에서 read로 불러오기에, site.txt를 편집할 때 마지막에 빈 라인 하나를 꼭 남겨두어야 한다.

Debian의 bash shell, macOS의 z shell에서 shell script의 작동이 확인되었다.

위와 거리가 먼 distro나 shell에서의 작동은 보장하지 않는다.

3-2. 작동 확인

```
ssu-20181755@ThinkPad-X200:~/NP/hw1$ ls
hw1_20181755.c  sites.txt  test_hw1_20181755.sh
ssu-20181755@ThinkPad-X200:~/NP/hw1$ sudo chmod 766 test_hw1_20181755.sh
[sudo] password for ssu-20181755:
ssu-20181755@ThinkPad-X200:~/NP/hw1$ ./test_hw1_20181755.sh

www.microsoft.com
- IPv4: 23.52.33.58
- IPv6: 2600:1410:4000:192::356e
- IPv6: 2600:1410:4000:1bb::356e

www.apple.com
- IPv4: 23.207.176.224
- IPv6: 2600:1410:1000:291::1aca
- IPv6: 2600:1410:1000:2b5::1aca

www.google.com
- IPv4: 142.250.76.132
- IPv6: 2404:6800:400a:813::2004

www.amazon.com
- IPv4: 23.50.1.154
- IPv6: 2600:1410:4000:191::3bd4
- IPv6: 2600:1410:4000:1b0::3bd4

www.gnu.org
- IPv4: 209.51.188.116
- IPv6: 2001:470:142:5::116

www.gnu.org
- IPv4: 209.51.188.116
- IPv6: 2001:470:142:5::116

www.debian.org
- IPv4: 130.89.148.77
- IPv4: 128.31.0.62
- IPv6: 2603:400a:ffff:bb8::801f:3e
- IPv6: 2001:67c:2564:a119::77

www.naver.com: 2024-
getaddrinfo IPv6: Name or service not known
- IPv4: 223.130.192.248
- IPv4: 223.130.192.247
- IPv4: 223.130.200.236
- IPv4: 223.130.200.219

www.daum.net
getaddrinfo IPv6: Name or service not known
- IPv4: 211.249.220.24
ssu-20181755@ThinkPad-X200:~/NP/hw1$ ./getaddrinfo www.qwerasdfzxcv.com
getaddrinfo IPv4: Name or service not known
getaddrinfo IPv6: Name or service not known
ssu-20181755@ThinkPad-X200:~/NP/hw1$ ls
ssu-20181755@ThinkPad-X200:~/NP/hw1$ ./test_hw1_20181755.sh
ssu-20181755@ThinkPad-X200:~/NP/hw1$
```

주소를 못 받아오는 internet protocol에 대하여 오류를 출력하고, 받아들 수 있는 internet protocol에 대하여 주소들을 출력한다.

미국 사이트들은 IPv6에 대한 지원이 훌륭하나, 내수외에는 사실상 쓰이지도 않는 국내 사이트들에서는 IPv6에 대한 지원을 찾아볼수가 없다. 네카라쿠베의 네카가 말이다.