Lab: Network Addressing

IC221, AY22S

Submission myhost.c

Test Script: ./test.sh

Task Description

In this lab, you will implement your own version of the Unix/Linux host command, to learn about the structures and functions used for addresss resolution during network socket programming in C. The Unix/Linux tool host performs a DNS look up of a domain name to resolve it to an IP address or addresses.

Address Info Structure. The Unix/Linux C standard library function to resolve a domain name to an IP address is getaddrinfo() and the structure used is called struct addrinfo.

struct addrinfo {

int ai\_flags;

int ai\_family;

int ai\_socktype;

int ai\_protocol;

size\_t ai\_addrlen;

struct sockaddr \*ai\_addr;

char \*ai\_canonname;

struct addrinfo \*ai\_next;

};

IP Address. The field ai\_addr stores the *socket address* (struct sockaddr), which we cast to an IPv4 *socket*, to eventually get the IP address. An interesting aspect of this structure is that it also a node within a linked list. The ai\_next field stores a pointer to the next addrinfo, which stores another resolved IP address for the domain. Eventually, ai\_next references NULL, which indicates the end of the list. We can iterate over the linked list like so:

struct addrinfo \* cur\_result, \*results, hints;

//...

//Convert the hostname to an address

if( (s = getaddrinfo(argv[1], NULL, &hints, &result)) != 0){

fprintf(stderr, "getaddrinfo: %s\n",gai\_strerror(s));

exit(1);

}

for(cur\_result = result; cur\_result != NULL; cur\_result = cur\_result->ai\_next){

//do something with the current result

}

TCP vs. UDP. Another important piece of addrinfo is the ai\_protocol, which describes how the domain name is resolved based on which application layer protocol we are interested in. For example, often domains have different IP addresses for web traffic and email, and this difference is represented in the ai\_protocol field. Since we are only interested in the primary resolution of the domain, we can check for that by comparing against the constant IPPROTO\_TCP since we want IP addresses that accept TCP connections.

cur\_result->ai\_protocol == IPPROTO\_TCP

IPv4 vs. IPv6. The last aspect of the addrinfo structure to consider is that ai\_family, which describes the kind of address was resolved. This could either be IPv4 (AF\_INET) or IPv6 (AF\_INET6). We are primarily concerned with IPv4, so we can compare ai\_family to AF\_INET to ensure we resolve right kind of IP address:

cur\_result->ai\_family == AF\_INET

For both of these options, we can specify the choice of ‘hint’ when making the getaddrinfo() request. Then getaddrinfo() will only return results that fit those requirements.

**Task** completemyhost.c

Requirements

* For a given domain name, resolve all the IPv4 addresses associated with ‘protocol 0’ for that domain.
* Print out the results as similarly to the Unix/Linux host command as possible.
* Print an error message to stderr if no domain name or a bad domain name are supplied

Sample output (IP addresses may vary):

$ ./myhost facebook.com

facebook.com has address 31.13.69.228

$ ./myhost microsoft.com

microsoft.com has address 191.239.213.197

microsoft.com has address 104.40.211.35

microsoft.com has address 104.43.195.251

$ ./myhost bad.bad.bad

getaddrinfo: Name or service not known

$ ./myhost

getaddrinfo: Name or service not known

EXTRA CREDIT (5 points): Add the ability for your myhost program to also resolve the IPv6 address with the same format as host. For example:

$ ./myhost facebook.com

facebook.com has address 31.13.69.228

facebook.com has IPv6 address 2a03:2880:f103:83:face:b00c:0:25de

$ ./myhost google.com

google.com has address 172.217.15.78

google.com has IPv6 address 2607:f8b0:4004:810::200e

Check out the man pages for inet\_ntop() and inet\_pton() for some useful details.