Internet Protocol Version 6

The version 6 of the IP protocol is designed to replace the current version 4 of the protocol.

Notes:

IP Addresses change from 32 bits to 128 bits.

There are backward compatibility features:
1) old addresses translate to new addresses: 96 bits of 0.
2) Dual stack host can send and receive both.
3) Dual stack router can forward both.
4) Dual protocol routers can translate packets from/to IPv4 to/from IPv6

The IPv6 definitions are already available in the `.h` files.

Almost all Linux machines are dual stack.
Activate by assigning IP4 and IP6 addresses.

IPv6 structures (in.h)

The address is 128 bits (was `sin_addr`).
In both versions it is stored inside a struct.
In version 4 it was a long so the nesting didn't matter.
In version 6 placing the array in a struct avoids the complications arising from the fact that arrays are represented by pointers.

```
struct in6_addr {
  u_int8_t  s6_addr[16];
}
```

In the version6 address structure, the family and port are the same as before, just newer type names for the same sizes. The family will be `AF_INET6`.
The flow information variable is new, exactly how it is to be used in not fully defined.
The addrress portion uses the new structure.

```
struct sockaddr_in6 {
  u_char          sin6_family;
  u_int16m_t      sin6_port;
  u_int32m_t      sin6_flowinfo;
  struct in6_addr sin6_addr;
};
```

There are version 6 value for the constants. (`in.h`)

Initializing the address structure is similar to version 4.

```
struct sockaddr_in6 sin6;
/* Server */
sin6.sin6_family = AF_INET6;
sin6.sin6_flowinfo = 0;
sin6.sin6_port = htons(23);
sin6.sin6_addr = in6addr_any;
/* Client with server on local host */
sin6.sin6_family = AF_INET6;
sin6.sin6_flowinfo = 0;
sin6.sin6_port = htons(23);
sin6.sin6_addr = in6addr_loopback;
```

The API calls take version 6 parameters

```
s = socket(AF_INET6, SOCK_STREAM, IPPROTO_IPV6);
if (bind(s,(struct sockaddr *)&sin6,sizeof(sin6))<0)
{
  errexit(...);
}
```

The socket options have version 6 equivalents.

There are a few additional options.

```
setsockopt(fd, IPPROTO_IPV6, IPV6_HOPLIMIT,
  &one, sizeof(one));
```

Names to Numbers

The system administrator can set up a machines Domain Name Service so that `gethostbyname` returns IPv6 addresses.

(Admin Detail) `/etc/resolv.conf` add the line

```
options inet6
```

Same call:

```
hp = gethostbyname(...);
```

Returned structure:
`hp->h_length` will be 16 (was 4)

Change the memcopy code to use a `sin6`

```
memcpy(&sin6.sin6_addr,hp->h_addr,hp->h_length);
```

Problem: support both IP4 and IP6 addresses

Solution: use `getaddrinfo` instead.

getaddrinfo

Designed as a generalization of (and improvement on) gethostbyname/addr and getservicebyname/port.

The following structure is used by getaddrinfo. It is used both to specify the nature of the request (IP4, IP6, ...) and to return the answer.

```
struct addrinfo {
 int ai_flags;
 int ai_family;
 int ai_sockettype;
 int ai_protocol;
 int ai_addrlen;
 struct sockaddr *ai_addr;
 char *ai_canonname;
 struct addrinfo *ai_next;
};
```

`ai_family`: `AF_INET` or `AF_INET6`
`ai_socktype`: `SOCK_STREAM` or `SOCK_DGRAM`
`ai_protocol`: The protocol to use.
`ai_flags`: control what is returned
`ai_addr`: the address returned
`ai_canonnname`: the fully qualified host name
`ai_next`: because sizes can vary a linked list is returned.

```
int getaddrinfo(const char *host,
                const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

`host`: The name or IP number of a host. IP number can be given in IP4 dotted notation, IP6 byte notation or IP6 mixed notation (see hint). If this is `NULL`, then the appropriate loop back address will be returned.

`service`: The name (eg. `"telnet"`) or number (eg. `"7654"`) of a service. If this is `NULL` the port number in the result will not be initialized.

`hints`: Specifies the type of lookup should be done.

`res`: The information being returned (result).

RETURN value: 0 if no error, otherwise the value is an error code that can be used with `gai_strerror` to get a string describing the error.

Details

`hints`:

`ai_family` specifies whether a IP4 or IP6 address should be returned. A value of `AF_UNSPEC` returns either or both (linked list).
`ai_socktype` UDP/TCP, 0 means retrieve all.
`ai_flags` what should be returned
`AI_CANONNAME` return the canonicanical name
`AI_ADDRCONFIG` used with unspec, return only addresses this host is configured to use. (Don't return an IPv6 if we have no IPv6 interfaces configured.)

`res`:

This is created by the `getaddrinfo` call. Consequently it needs to be released using a `freeaddrinfo` call if you are doing a bunch of get addrs. The `ai_addr` field is an entire address structure, including the socket type, port number and address; for the appropriate family (IP4/6).

A linked of list structures is returned if more than one address matches the hints.

strings to/from addresses

New calls are needed to turn a string into an address or an address into a string

String to address:

`inet_pton` Turn a string into an address

```
inet_pton(AF_INET6,"::134.139.248.19",&a.sin6_addr);
inet_pton(AF_INET6,"::8f8b:f813",&a.sin6_addr);
```

The `::` indicates 0's.
`a` must be a `sockaddr_in6`

Address to string:

`inet_ntop` Turn an address into a string

```
inet_ntop(AF_INET6,infoptr->ai_addr,b,sizeof(b));
```

`b` should be a character array of size at least `INET6_ADDRSTRLEN`.

Primitive Client

```
int main() {
  int s;
  char message[80];
  struct sockaddr_in6  srv_addr;
  s = socket(PF_INET6, SOCK_STREAM, 0);
  memset(&srv_addr, 0, sizeof(srv_addr));
  inet_pton(AF_INET6,"::134.139.248.19",
    &srv_addr.sin6_addr);
  srv_addr.sin6_family = AF_INET6;
  srv_addr.sin6_port = htons(7654);
  connect(s, (struct sockaddr *) &srv_addr,
    sizeof(srv_addr));
  strcpy(message,"Client speaks");
  write(s, message, 80);
  read(s, message, 80);
  close(s);
  return 1;
}
```

Primitive Client Notes

Changes:

1) lots of "6"s

2) used `inet_pton`

Primitive Client 2

```
int main() {
  int s;
  char message[80];
  char addr_buf[INET6_ADDRSTRLEN]; /* for ntop */
  struct sockaddr_in6  srv_addr;
  struct addrinfo hint, *infoptr;
  int result;
  hint.ai_family = AF_INET6;
  hint.ai_socktype = SOCK_STREAM;
  hint.ai_protocol = 0; /* any protocol */
  hint.ai_flags = AI_CANONNAME | AI_ADDRCONFIG;
  result = getaddrinfo("puma",NULL,&hint,&infoptr);
  memcpy(&svr_addr,infoptr->ai_addr,
    infoptr->ai_addrlen);
  srv_addr.sin6_port = htons(7654);
  s = socket(PF_INET6, SOCK_STREAM, 0);
  connect(s, (struct sockaddr *) &srv_addr,
    sizeof(srv_addr));
  strcpy(message,"Client speaks");
  write(s, message, 80);
  read(s, message, 80);
  close(s);
  return 1;
}
```

Primitive Client 2 Notes

Changes:

1) lots of "6"s

2) used `getaddrinfo`

3) mem copied the entire structure, not just the address part.

Could have:

```
getaddrinfo("puma","7654",&hint,&infoptr);
```

instead of doing it separately

Primitive server

```
int main(){
  int master, client, len; char message[80];
  struct sockaddr_in6 my_addr, his_addr;
  /* get a socket allocated */
  master = socket(PF_INET6, SOCK_STREAM, 0);
  /* bind to the well-known port on our machine */
  memset(&my_addr, 0, sizeof(my_addr));
  my_addr.sin6_family = AF_INET6;
  my_addr.sin6_flowinfo = 0;
  my_addr.sin6_addr = in6addr_any;
  my_addr.sin6_port = htons(7654);
  bind(master, (struct sockaddr *) &my_addr,
    sizeof(my_addr));
  listen(master, 5);
  len=sizeof(his_addr);
  /* get the connection to the client */
  client = accept(master,
    (struct sockaddr *) &his_addr, &len);
    /* get the message from the client */
  read(client,message,80);
  strcpy(message,"Server replies");
  write(client, message, 80); /* send reply */
  close(client);
  close(master);
  return 1;
}
```

Primitive Server Notes

Changes:

1) lots of ”6”s

2) `in6addr_any`