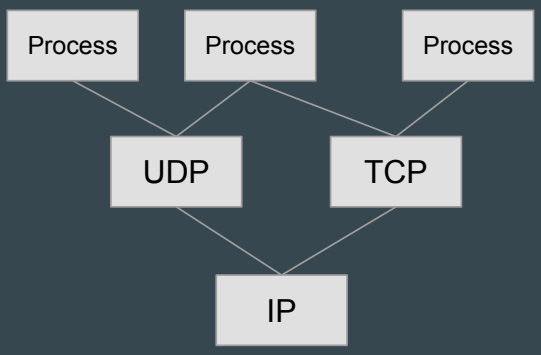
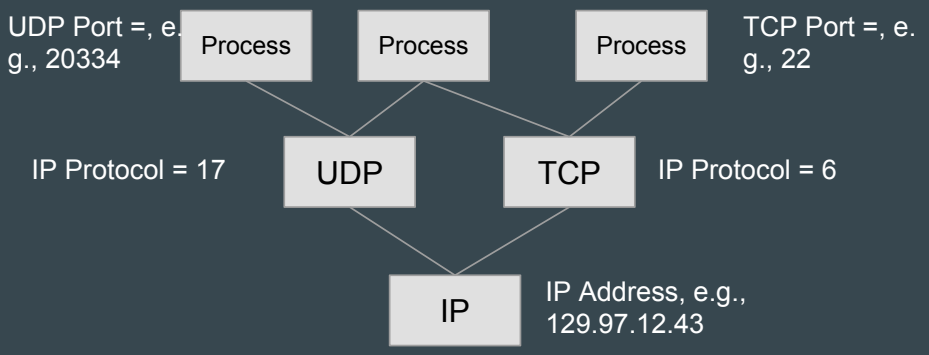
Focus on UDP & TCP over IP

- UDP & TCP are transport-layer protocols
- Over IP, which is a network-layer protocol



- UDP & TCP multiplexed over IP.
- Multiple processes multiplexed over each of UDP & TCP.

Multiplexing



The “5-tuple”

- On an (the) Internet, a connection or association is identified uniquely by the 5-tuple:
  - ⟨source-ip-address, source-port, destination-ip-address, destination-port, protocol⟩
  - E.g., ⟨129.197.2.13, 20334, 216.58.199.14, 80, 6⟩
  - E.g., ⟨129.197.2.13, 50000, 216.58.199.14, 80, 6⟩

1 5 Tuple

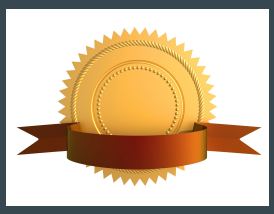
We can understand the flow of the packet through your system by looking at the 5-tuple.

- source ip address
- source port
- destination ip address
- destination port
- protocol

The software view

- OS/library for UDP/TCP
- Applications can be written on top of UDP/TCP
- Client (initiator) - server (responder) paradigm.

The socket API



- POSIX standard API for UDP/TCP applications
- (Does not necessarily mean that it's a great API.)
- Essential calls:

  - *socket()*
  - *listen()*
  - *accept()*
  - *send()*
  - *sendto()*
  - *bind()*
  - *connect()*
  - *receive()*
  - *sendto()*
  - *recvfrom()*

2 Socket API

Here a the 9 basic socket commands.

Here is a UDP example: You have multiple identities association with ip so you can just chose one or use all of them.

Checkout `man 7 ip` for more information.

```
struct in_addr // this is a struct for internet addresses, it is just a uint32

int sockfd = -1
if(sockfd == socket(AF_INET, SOCK_DGRAM, 0) < 0) { // see man socket to learn how this
    function works
   //shit done borked
   return -1
}

struct sockaddr_in server
server.sin_port = 0 // some ports have specific privileges and such, 0 allows the OS to
    pick the port
server.sin_port_family = AF_INET
if(bind(sockfd, (struct aockaddr_in *)&server, sizeof(struct socaddr_in)) < 0) { // see
    man bind
   //what did you do wrong
   return -1
}

if(getsockname(sockfd, (struct sockaddr) &server, sizeof(struct socaddr_in)) < 0){ //man
    getsockname to get the socket address
   //seriously how did you fuck this up
   return -1
}

print(inet_ntoa(server.sin_addr))
print(ntohs(server, sin_port))

//make sure you put that buffer length thing properly (see buffer overflow attacks, hee
    hee)
//this is blocking which is why you have to check the reclen, might actually need a loop
    here
if(reclen = recfrom(sockfd, buffer, bufferlength-1, 0, (struct socketaddr *) &client,
    sizeof(struct sockaddr_in)) < 0) { // man recvfrom
   //layyyyyme
   return -1;
}

print(inet_ntoa(client.sin_addr))
print(ntohs(client, sin_port))

//dont forget to close your shit
close(sockfd)
```

```
//To get the address you want
getifaddrs
//avoid memory leaks
freeifaddrs

struct ifaddrs *ifa //this is a linked list
if(getifaddrs(&ifa) < 0){
   return -1
}

for(struct ifaddrs *1 ifa; i != NULL; i = i->ifa_next) {
   if(i->ifa_addr == NULL) continue

   //add address to list of available addresses that the user can use
}
```

We need to watch for big/little endian cause OSs are separate from networks. We have a call in `in_addr` that will fix its shit for us.

CHECK YOUR RETURN VALUES FUCKER!!! This shit sucks to debug. LARA I'M LOOKING AT YOU, IF YOU FUCK THIS UP I WILL KILL YOU.

Helpful commands:

- man all the pages
- netstat