Chapter 14
Multiple Protocol Servers

Motivation: Separate servers take more RAM and, because of context switches, more CPU. One server handling both UDP and TCP connections is efficient.

OS support: UDP and TCP are separate connections even on the same port.

Implementation: Open one socket for TCP and one *on the same port* for UDP. Use `select` to determine which one needs service.

Example: The daytime server has both UDP and TCP sockets.

A TCP client does a "connect", which is handled by an accept, write, close.

A UDP client does a "send", which is handled by a receive-from, send-to.

Detailed Analysis

See section 14.5 for code.

```
int main(int argc,char *argv[]){
  char *service="daytime";
  char buf[LINELEN+1];
  struct sockaddr_in fsin;
  unsigned int alen;
  int tsock;
  int usock;
  int nfds;
  fd_set rfds;
  switch (argc){
   ...
  }
  tsock=passiveTCP(service,QLEN);
  usock=passiveUDP(service);
```

`tsock`: is the TCP master socket
`usock`: is the UDP socket
`nfds`: the size of the set
`rfds`: the set, instead of a copy, it is rebuilt each time.
Both the TCP and UDP sockets are set up

The `switch` and other declarations are identical to other servers.

```
nfds=MAX(tsock,usock)+1;
FD_ZERO(&rfds);
while(1) {
  FD_SET(tsock,&rfds);
  FD_SET(usock,&rfds);
  if (select...)
    errexit(...);
```

`nfds` is set to be as small as possible.

The set is constructed by the two `FD_SET` calls.
(There is no `afds` and no `memcpy`)
The `FD_ZERO` is outside the loop because, nothing ever turns on any `FD_SET` bits. (`select` turns some bits off).

The `select` call is unchanged.

```
if (FD_ISSET(tsock,&rfds) {
  int ssock;
  alen = sizeof(fsin);
  ssock=accept(tsock,
          (struct sockaddr *)&fsin,&alen);
  if (ssock < 0)
    errexit(...);
  daytime(buf);
  (void)write(ssock,buf,strlen(buf),0
          (struct sockaddr *)&fsin,sizeof(fsin));
  close(ssock);
}
```

Activity on the `tsock` means a client has done a `connectTCP`.

The code is the standard TCP daytime serer code; accept, get the time, write the string back.

The slave socket needs to be closed because the server is done with the client.

```
if (FD_ISSET(usock,&rfds) {
  alen = sizeof(fsin);
  if (recvfrom(usock,buf,sizeof(buf),0,
        (struct sockaddr *)&fsin, &alen)<0)
    errexit(...)
  daytime(buf);
  (void)sendto(usock,buf,strlen(buf),0
          (struct sockaddr *)&fsin,sizeof(fsin));
}
```

Activity on the `usock` means a client has done a `connectUDP` and a `write`.

The code is the standard UDP daytime server code; receive from, get the time, send the time back to the return address

Behavior

If a client does a `connectTCP`:

Input will arrive on `tsock`
Server will do an accept.

If a client does a `write` or `send` (after a `connectUDP`):

Input will arrive on `usock`
Server will do a recvfrom.

In both cases this server replies.

Extensions

The above code works only for short requests.
For long requests; possibly code similiar to Chapter 13 will do.

Consider echo. The interaction is long (until EOF keyboard).
We can do UDP receive from/send to for many clients.
We don't know how many clients there are.
We can use the `afds`, `memcpy` code for handling multiple TCP clients found in Chapter 13.
(Don't forget to put the `usock` in the `afds`.)