Chapter 10
Iterative Connection-Oriented Server

see Section 10.5 for code

```
int main(int argc, char *argv[]) {
  struct sockaddr_in fsin;
  char *service = "daytime";
  int msock, ssock;
  unsigned int alen;
  switch (argc)
  {
    case 1: break;
    case 2: service=argv[1]; break;
    default: errexit(...);
  }
  msock=passiveTCP(service,QLEN);
```

`fsin` will be the client's address (unused)

The default service is daytime.
If we have an argument (`argc==2`)
use it to override the default service.

The `bind` and `listen` are on the master socket.
`msock`: is the master socket.
Its TCP purpose is to produce slave sockets.

```
while (1){
  alen=sizeof(fsin);
  ssock=accept(msock,
    (struct sockaddr *)&fsin,&alen);
  if (ssock < 0) errexit("...");
  TCPdaytimed(ssock);
  close(ssock);
}
```

The server waits for a `connect` request using the master socket.

When a connect request arrives; `accept`: accepts a connection and creates a socket (`ssock`) for that connection.

A slave socket provides communication to *one* client.

`ssock` will be used by the server to communicate with the client.

The error exit behaviour on a failed accept is debatable. Most failure of accept indicate operating system problems and are not recoverable.

The service is encapsulated in a function call.
That call is passed the socket so it can talk to the client.

When the service is done, the slave socket is closed.

```
void TCPdaytimed(int fd) {
  char *pts
  time_t now;
  char *ctime();
  (void) time(&now);
  pts=ctime(&now);
  (void)write(fd,pts,strlen(pts));
}
```

Note: `fd` is the slave socket.

For this service we do not need a read.

We get the time using a system call.
We convert it to a string using a system call.
We send the string

Any message the client sends is ignored.

Writing to a socket closed on the other end either gets an error (TCP) or the message is quietly discarded (UDP).

In Comer's code, a client that exits early will cause an error that aborts the server. This behavior is bad. To avoid this you can either:
`(void)send(fd,pts,strlen(pts),MSG_NOSIGNAL);`
or in the main:
`signal(SIGPIPE, SIG_IGN);`
We will defer this problem to a later chapter.

Closing Sockets

Because in the main program the slave socket is closed; the connection between the client and the server is ended, the socket is deallocated and the descriptor is set to null. the next slave socket created may use the same slot in the descriptor array.

The server should close the master socket only when (if) the server exits.

Closing the master socket means no further requests for a connection will be accepted by the server.

Closing a slave socket means the server no longer wants to talk to the client.
A connection can still be made to a new client.

Note: if the program exits, all descriptors, including sockets, are automatically closed. Comer uses this behavior and does not close the master socket.
We recommend an explicit close.

Review

`ssock` is the connection between the server and a client.

`ssock` is the connection between an internet address/port pair of the server and an internet address/port pair of the client.

client view: `connect` attaches you to a socket of the server *created* by the address/port you named in the connect. It does not connect you to the master socket on the port you named.

UDP: The only (passive) socket is used to send and receive messages.

TCP: Master socket is used to `accept` which creates a new socket. The new (slave) socket is used for reads and writes.

Termination (Issue)

Single reply then terminate strategy:
Socket and structures will go away shortly.
Brief retention to ensure late packets are rejected gracefully.

Problem: if the server `while(1)`'s fast (due to alot of requests) the number of sockets waiting to go away can become large.

Design principle: consider the rate of requests.

Frequent programming problem: a failure triggers an immediate retry.