

```
pid_t pid;
int listenfd, connfd;
listenfd = Socket(...);

/* fill in sockaddr_in() with server's well-known port */
Bind(listenfd, ...);
Listen(listenfd, LISTENQ);

for ( ; ; ) {
    connfd = Accept(listenfd, ...); /* probably blocks */
    if ( (pid = Fork()) == 0 ) {
        Close(listenfd); /* child closes listening socket */
        doIt(connfd); /* process the request */
        Close(connfd); /* done with this client */
        exit(0); /* child terminates */
    }
    Close(connfd); /* parent closes connected socket */
}
```

Figure 4.13 Outline for typical concurrent server.

this file or socket. In Figure 4.13, after socket returns, the file table entry associated with `listenfd` has a reference count of 1. After `accept` returns, the file table entry associated with `connfd` has a reference count of 1. But after `fork` returns, both descriptors are shared (i.e., duplicated) between the parent and child, so the file table entries associated with both sockets now have a reference count of 2. Therefore, when the parent closes `connfd`, it just decrements the reference count from 2 to 1 and that is all. A real close on the descriptor does not take place until the reference count reaches 0. This will occur at some time later when the child closes `connfd`.

We can also visualize the sockets and the connection that occurs in Figure 4.13 as follows. First, Figure 4.14 shows the status of the client and server while the server is blocked in the call to `accept` and the connection request arrives from the client.

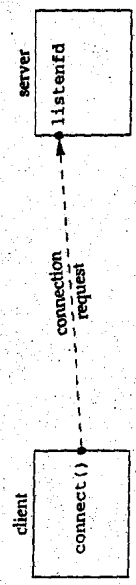


Figure 4.14 Status of client-server before call to `accept` returns.

Immediately after `accept` returns we have the scenario shown in Figure 4.15. The connection is accepted by the kernel and a new socket, `connfd`, is created. This is a connected socket and data can now be read and written across the connection.

The next step in the concurrent server is to call `fork`. Figure 4.16 shows the status after `fork` returns.

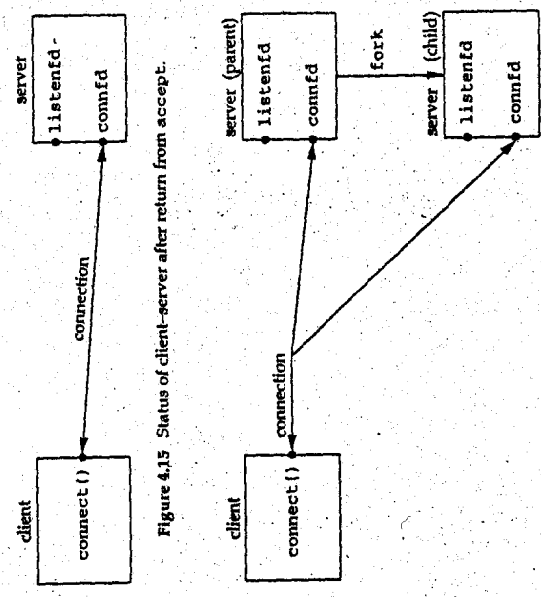


Figure 4.15 Status of client-server after return from `accept`.

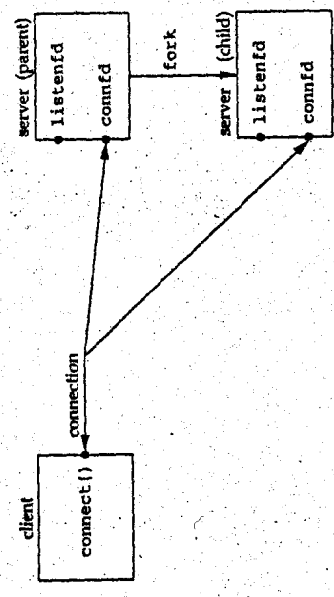


Figure 4.16 Status of client-server after `fork` returns.

Notice that both descriptors, `listenfd` and `connfd`, are shared (duplicated) between the parent and child.

The next step is for the parent to close the connected socket and the child to close the listening socket. This is shown in Figure 4.17.

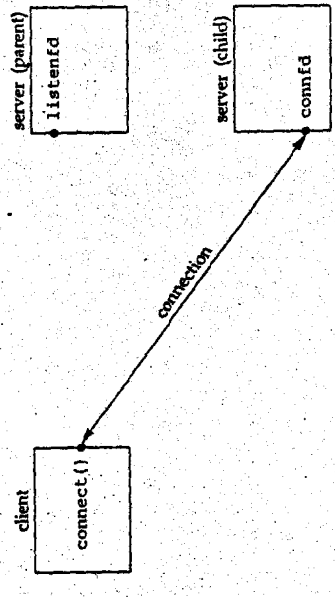


Figure 4.17 Status of client-server after parent and child close appropriate sockets.

This is the desired final state of the sockets. The child is handling the connection with the client and the parent can call `accept` again on the listening socket, to handle the next client connection.