Handling Multi-Client on Server

Source Code: *URL* shorturl.at/cekW7

In a multi client chat server, *N* clients are connected to a server and send messages. In this program, one of the clients send messages to the server and it will send back the messages to all other clients. The code is implemented using C language, with a TCP socket connection.

client_multi.c

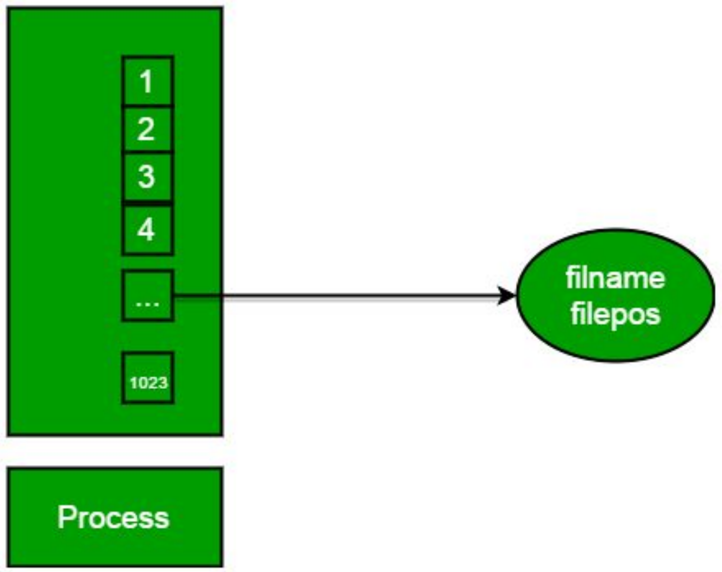
server_multi.c

File Descriptor

- What is the File Descriptor?
  - File descriptor is integer that uniquely identifies an open file of the process.
- File Descriptor table:
  - File descriptor table is the collection of integer array indices that are file descriptors in which elements are pointers to file table entries. One unique file descriptors table is provided in operating system for each process.

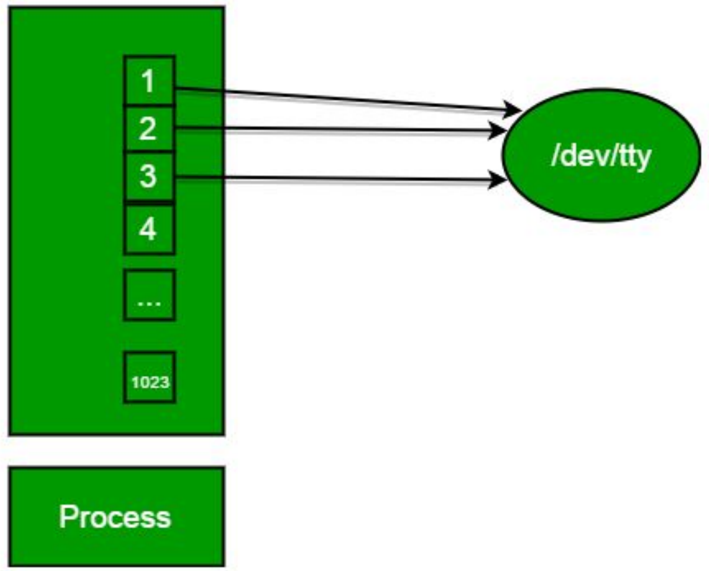
Contd..

- **File Table Entry:**
  - **File table entries is a structure In-memory surrogate for an open file, which is created when process request to opens file and these entries maintains file position.**



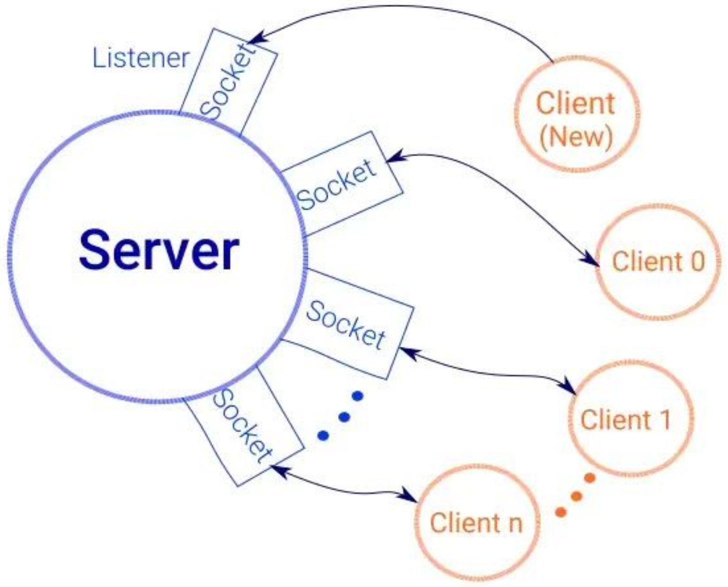
Contd..

- **Standard File Descriptors:**
  - **When any process starts, then that process file descriptors table's fd(file descriptor) 0, 1, 2 open automatically, (By default) each of these 3 fd references file table entry for a file named** /dev/tty



Sockets and concurrency

- Sockets are treated just like files:
  - socket() returns a file descriptor
  - read() & write()
    - same interface as for files



Handling Connections

- ❑ Examples of network applications that rely on handling concurrent connections
  - ➢ Web servers, file servers, IRC, IM, etc…
- ❑ How can an application handle multiple connections
  - ➢ Threading & select()

Select()

- How select works
  - Allows you to monitor multiple sockets
  - API for monitoring multiple file descriptors
- Two main sets of descriptors
  - Read & write descriptors
- Use bit-array fd_set to monitor
  - fd_set readfds, writefds;
- First set all to 0
  - FD_ZERO(&readfds); FD_ZERO(&writefds);

#include <sys/select.h>

```
int select(int nfds,
fd_set *readfds,
fd_set *writefds,
fd_set *exceptfds,
struct timeval
*timeout);
```

Telling Select What To Monitor

- After zeroing out, set FDs to be monitored
  - Original FD returned by socket() for incoming connections
  - All currently connected client’s FDs
- So, assuming sfd=socket(…);
  - FD_SET(sfd, &readfds);
  - Loop through client FDs:
    - FD_SET(client[i].sfd, &readfds)

Using select()

- ❑ Select checks from bit 0 in the bit-array up until maxfd
  - ➢ Initially: maxfd=sfd;
  - ➢ Looping through clients: if(client[i].fds>maxfd) …
- ❑ Now, call it!
  - ➢ select(maxfd+1, &readfds, NULL, NULL, NULL);

IMPORTANT: select() overwrites &readfs with new bit-array, representing which file descriptors are ready

After Select() Returns

- Check the new bit-array
- What if select() sets the bit for sfd?
  - You have a new client
  - if (FD_ISSET(sfd, &readfds)) { accept_client(…); }
- What if select() sets the bit for a client FD?
  - Data is ready to be read
  - Must loop through all of your client FDs with FD_ISSET

Handle the whole array...

- After you've gone through and used *FD_ISSET*():
  - Start all over!
  - *FD_ZERO*(&readfds);
  - *FD_SET*(sfd, &readfds);

- In other words, create a while(1) around this and loop and loop!

More on Accepting New Clients

- ❑ Whenever FD_ISSET(sfd, &readfds), you have a new client
- ❑ You use accept() on sfd and save the returned file descriptor as your new client's FD
- ❑ Store all of these FDs and make sure to set them in readfds before you call select()

Thank You