NETWORK PROGRAMMING LINUX SOCKET PART 2: THE SERVER SIDE ISSUES

My Training Period: xx hours

Note: Program examples if any, compiled using gcc on **Linux Fedora Core 3** machine with several update, as normal user. The Fedora machine used for the testing having the "No Stack Execute" disabled and the SELinux set to default configuration.

The Client-Server Model

- TCP/IP enables peer-to-peer communication.
- Computers can cooperate as equals or in any desired way.
- Most distributed applications have special roles. For example:

  1. Server waits for a client request.
  2. Client requests a service from server.

Some Security Definitions

- Authentication: verifying a computer's identity.
- Authorization: determining whether permission is allowed.
- Data security: preserving data integrity.
- Privacy: preventing unauthorized access.
- Protection: preventing abuse.
- These security matters have experienced quite a pretty good evolution. The standards also have been produced such as the obsolete C2, formally known as Trusted Computer System Evaluation Criteria (TCSEC) (PDF format) then superseded by Common Criteria and the ISO version: ISO 15408 Common Criteria for Information Technology Security Evaluation (another reference can be found at commoncriteriaportal.org).

Connectionless (UDP) vs Connection-Oriented (TCP) Servers

- Programmer can choose a connection-oriented server or a connectionless server based on their applications.
- In Internet Protocol terminology, the basic unit of data transfer is a datagram. This is basically a header followed by some data. The datagram socket is connectionless.

Menu

Network Story 1
Network Story 2
Network Story 3
Network Story 4
Network Story 5
Network Story 6
Socket Example 1
Socket Example 2
Socket Example 3
Socket Example 4
Socket Example 5
Socket Example 6
Socket Example 7
Advanced TCP/IP 1
Advanced TCP/IP 2
Advanced TCP/IP 3
Advanced

TCP/IP 4
Advanced TCP/IP 5

- User Datagram Protocol (UDP):

  1. Is a connectionless.
  2. A single socket can send and receive packets from many different computers.
  3. Best effort delivery.
  4. Some packets may be lost some packets may arrive out of order.

- Transmission Control Protocol (TCP):

  1. Is a connection-oriented.
  2. A client must connect a socket to a server.
  3. TCP socket provides bidirectional channel between client and server.
  4. Lost data is re-transmitted.
  5. Data is delivered in-order.
  6. Data is delivered as a stream of bytes.
  7. TCP uses flow control.

- It is simple for a single UDP server to accept data from multiple clients and reply.
- It is easier to cope with network problems using TCP.

Stateless vs Stateful Servers

- A stateful server remembers client data (state) from one request to the next.
- A stateless server keeps no state information.

- Using a stateless file server, the client must:

  1. Specify complete file names in each request.
  2. Specify location for reading or writing.
  3. Re-authenticate for each request.

- Using a stateful file server, the client can send less data with each request.
- A stateful server is simpler.
- On the other hand a stateless server is:

  1. More robust.
  2. Lost connections can't leave a file in an invalid state.
  3. Rebooting the server does not lose state information because there is no state information hold.
  4. Rebooting the client does not confuse a stateless server.

Concurrent Processing

Concurrency

- Real or apparent simultaneous processing.
- Time-sharing: a single CPU switches from 1 process to the next.
- Multiprocessing: multiple CPUs handle processes.

Network Concurrency

- Multiple distributed application share a network.
- With a single Ethernet segment or hub, one packet at a time can use the network. Network sharing is like time-sharing.
- With a switch, multiple packets can be in transit and transfer simultaneously, like multiprocessing.
- With several networks, multiple packets can be in transit and transfer like having multiple computers.

Server Concurrency

- An iterative server finishes one client request before accepting another.
- An iterative telnet daemon is almost useless.
- Concurrent servers are difficult to write.
- We will consider several designs for concurrency.

Programs vs Processes

- Program: executable instructions.
- Process: program being executed.
- Each process has its own private data.

  e.g.: Multiple `pico` processes have different text on the screen.

Concurrency using fork() in UNIX/Linux

```
/* testpid.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main (int argc, char **argv)
{
    int i, pid;
    pid = fork();

    printf("Forking...the pid: %d\n", pid);
    for (i = 0; i < 5; i++)
        printf(" %d    %d\n", i, getpid());
    if (pid)
       wait(NULL);
   return 0;
}
```

```
[bodo@bakawali testsocket]$ gcc -g testpid.c -o testpid
[bodo@bakawali testsocket]$ ./testpid
```

```
Forking...the pid: 0
 0    27166
 1    27166
 2    27166
 3    27166
 4    27166
Forking...the pid: 27166
 0    27165
 1    27165
 2    27165
 3    27165
 4    27165
```

- New process starts execution by returning from fork().
- Child and parent are nearly identical.
- Parent gets the child process id from fork().
- Child gets 0 back from fork().
- Parent should wait for child to exit.

Time slicing

- Round robin technique.
- Operating system is interrupted regularly by a clock.
- In the clock interrupt handler the kernel checks to see if the current process has exceeded its time quantum.
- Processes are forced to take turns using the CPU but every process will get their time slice.

Using exec family to execute a new program in UNIX/Linux

- The following are exec family prototypes.

```
int execve(const char *file, char *const argv [ ], char *const
envp[ ]);
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char * const
envp[ ]);
int execv(const char *path, char *const argv[ ]);
int execvp(const char *file, char *const argv[ ]);
```

- exec family executes a new program (same process id).
- The first argument is the new program if it says path, it requires a full pathname otherwise, and it searches in the current path.
- Program arguments follow as a list or a vector.
- execve() and execle() allow specifying the environment.

Context Switching

- Having multiple server processes means context switches.
- A context switch consumes CPU time.
- Other processes are blocked during that time.
- We must consider the benefits of multiple servers versus the overhead.

Asynchronous I/O

- Asynchronous (instead of synchronous) I/O means allowing a process to start an I/O operation and proceeding with other work while the I/O occurs. Another term used is non-blocking (instead of blocking).

- Obviously, asynchronous/non-blocking provides more robust and efficient way for concurrent connection.
- UNIX I/O occurs asynchronously if you use select().
- A process asks the select() system call to tell which of a collection of file descriptors is ready to finish I/O.
- After calling select() the process can call read() or write() to perform I/O which is at that time no more than a copying of data to/from kernel space with real I/O either already done or scheduled for later.
- A server process can use select() to determine which of a collection of sockets it can read without blocking.

Programming Interfaces

TCP/IP Application Programming Interface (API)

- API: routines supplied by the OS defining the interface between an application and the protocol software.
- Better to avoid vendor-specific data format and features, use standard APIs for portability.
- The API only suggests required functionality and it depend on the implementation.
- For UNIX - socket (original Berkeley system calls) and TLI (Transport Layer Interface - AT&T UNIX System V).
- For Apple Mac – MacTCP.
- For MS Windows – Winsock (still based on the Berkeley socket) and Winsock in .NET.
- There is also other TCP/IP APIs that implementation dependent.
- Unices TCP/IP APIs are kernel system calls.
- Mac and Windows using extension/drivers and dynamic link library (dll).
- In this Tutorial we will use socket APIs and in general socket refer to socket APIs that includes the socket().
- In fact the APIs just routines/functions in C language.

Required Functionality

- Allocate resources for communication such as memory.
- Specify local and remote endpoints.
- Initiate a client connection.
- Wait for a client connection.
- Send or receive data.
- Determine when data arrives.
- Generate urgent data.
- Handle received urgent data.
- Terminate a connection.
- Abort.
- Handle errors.
- Release resources.

System Calls

- An operating system should run user programs in a restricted mode i.e. user program should not do I/O directly.
- User programs should make a system call to allow trusted code to perform I/O.
- In UNIX, functions like open(), read(), write(), close() are actually system calls.
- A UNIX system call is a transition from user mode to kernel mode.
- TCP/IP code is called through the system calls.

UNIX I/O with TCP/IP

- To a certain degree, I/O with sockets is like file I/O.
- TCP/IP sockets are identified using file descriptors.
- read() and write() work with TCP/IP sockets.
- open() is not adequate for making a connection.
- Calls are needed to allow servers to wait for connections.
- UDP data is always a datagram and not a stream of bytes.

The Socket Interface

- Socket is an Application Programming Interface (API) used for Interprocess Communications (IPC).
- It is a well defined method of connecting two processes, locally or across a network.
- It is a Protocol and Language Independent.
- Often referred to as Berkeley Sockets or BSD Sockets.
- The following figure illustrates the example of client/server relationship of the socket APIs for connection-oriented protocol (TCP).

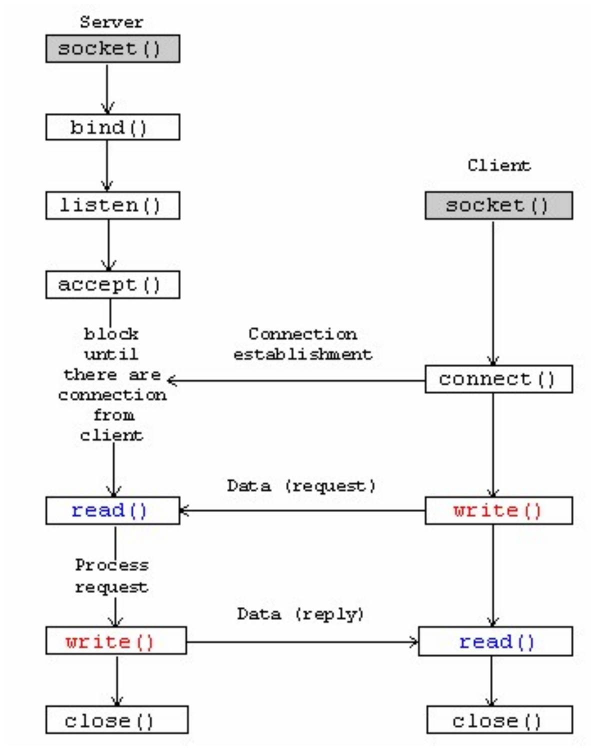

Figure 8

- The following figure illustrates the example of client/server relationship of the socket APIs for a connectionless protocol (UDP).

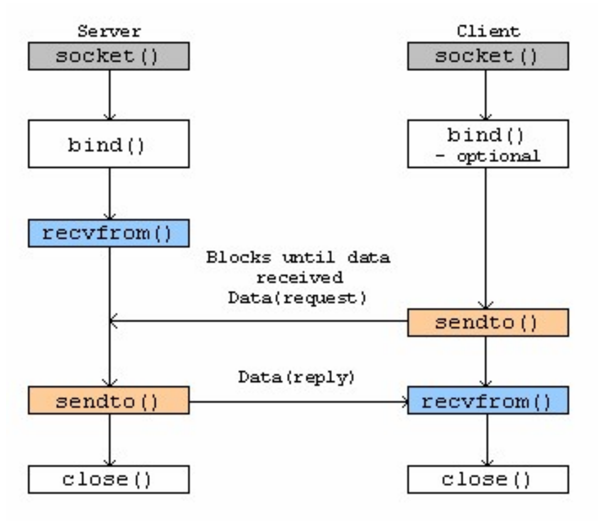

Figure 9

Berkeley Sockets: API for TCP/IP Communication

- Uses existing I/O features where possible.
- Allows TCP/IP connections, internal connections, and possibly more.
- Started in BSD UNIX in the 1980s.
- BSD UNIX adopted by Sun, Tektronix and Digital.
- Now the socket interface is a de facto standard.
- Sockets make the network look much like a file system.

Socket Descriptors

- UNIX open() yields a file descriptor: a small integer used to read/write a file.
- UNIX keeps a file descriptor table for each process an array of pointers to the data about the open files.
- A file descriptor is used to index the array.
- Sockets are added to this abstraction.
- The socket() system call returns a socket descriptor.
- Actually, files and sockets are accessed using the same table.
- The structure pointed to by a table entry has a field which tells whether it is a file or socket.

System Data Structures for Sockets

- In order to use a socket, the kernel needs to keep track of several pieces of data as the following:

    1. Protocol Family: a parameter to the socket call.
    2. Service Type (Stream, Datagram): parameter to socket.
    3. Local IP Address: can be set with bind().
    4. Local Port: can be set with bind().
    5. Remote IP Address: can be set with connect().
    6. Remote Port: can be set with connect().

- Ultimately all 6 values must be known to 'make' the communication.

Active vs Passive Sockets

- A server uses a passive socket to wait the client connections.
- A client uses an active socket to initiate a connection.
- Both start using the socket() call.

- Later on, servers and clients will use other calls.

Socket Endpoints

- TCP/IP communication occurs between 2 endpoints.
- An endpoint is defined as an IP address and a port number.
- To allow other protocols to merge into the socket abstraction, address families are used.
- We will use PF_INET for internet protocol family.
- We will also use AF_INET for internet address family. Normally PF_INET = AF_INET = 2.
- Socket types for AF_INET are listed in the following Table.

Socket type	Protocol	TYPE
STREAM	TCP, Systems Network Architecture (SNA-IBM), Sequenced Packet eXchange (SPX-Novell).	SOCK_STREAM
SEQPACKET	SPX.	SOCK_SEQPACKET
DGRAM	UDP, SNA, Internetwork Packet eXchange (IPX-Novell).	SOCK_DGRAM
RAW	IP.	SOCK_RAW

Table 4: AF_INET socket combinations

- There are other address families that you will find somewhere and sometime such as:

  1. AF_UNIX
  2. AF_NS
  3. AF_TELEPHONY

AF_UNIX address family

- The system uses this address family for communicating between two programs that are on the same physical machine. The address is a path name to an entry that is in a hierarchical file system.
- Sockets with address family AF_UNIX use the sockaddr_un address structure:

```
struct sockaddr_un {
short sun_family;
char sun_path[126];
};
```

- The sun_family field is the address family. The sun_path field is the pathname. The <sys/un.h> header file contains the sockaddr_un address structure definition.
- For the AF_UNIX address family, protocol specifications do not apply because protocol standards are not involved.
- The communications mechanism between the two processes on the same machine is specific to that machine.

AF_NS address family

- This address family uses addresses that follow Novell or Xerox NS protocol definitions. It consists of a 4-byte network, a 6-byte host (node), and a 2-byte port number.
- Sockets with address family AF_NS use the sockaddr_ns address structure:

```
struct   sockaddr_ns {
unsigned short   sns_family;
struct ns_addr   sns_addr;
char sns_zero[2];
};
```

AF_TELEPHONY address family

- Telephony domain sockets (sockets that use the AF_TELEPHONY address family) permit the user to initiate (dial) and complete (answer) telephone calls through an attached ISDN telephone network using standard socket APIs.
- The sockets forming the endpoints of a connection in this domain are really the called (passive endpoint) and calling (active endpoint) parties of a telephone call.
- The AF_TELEPHONY addresses are telephone numbers that consist of up to 40 digits (0 - 9), which are contained in sockaddr_tel address structures.
- The system supports AF_TELEPHONY sockets only as connection-oriented (type SOCK_STREAM) sockets.
- Keep in mind that a connection in the telephony domain provides no more reliability than that of the underlying telephone connection. If guaranteed delivery is desired, you must accomplish this at the application level, such as in fax applications that use this family.
- Sockets with address family AF_TELEPHONY use the sockaddr_tel address structure.

```
struct sockaddr_tel {
short stel_family;
struct tel_addr stel_addr;
char stel_zero[4];
};
```

- The telephony address consists of a 2-byte length followed by a telephone number of up to 40 digits (0 - 9).

```
struct tel_addr {
unsigned short t_len;
char t_addr[40];
};
```

- The stel_family field is the address family. The stel_addr field is the telephony address, and stel_zero is a reserved field. The <nettel/tel.h> header file contains the tel_addr and sockaddr_tel structure definitions.

*Continue on next Module…*More in-depth discussion about TCP/IP suite is given in Advanced TCP/IP Programming Tutorials.

Further reading and digging:

1. Check the best selling C/C++, Networking, Linux and Open Source books at Amazon.com.
2. Protocol sequence diagram examples.
3. Another site for protocols information.
4. RFCs.
5. GCC, GDB and other related tools.

| Winsock & .NET | Winsock | < Socket Programming Introduction | Linux Socket Index | Socket Programming Interface (APIs) > |