UNIX domain sockets

Troy D. Hanson
<tdh@tkhanson.net>

Communication within a host

UNIX domain sockets are a method by which processes on the same host can communicate. Communication is bidirectional with stream sockets and unidirectional with datagram sockets.

```
fd = socket(AF_UNIX, SOCK_STREAM, 0);
```

Identity

Instead of identifying a server by an IP address and port, a UNIX domain socket is known by a pathname. Obviously the client and server have to agree on the pathname for them to find each other. The server binds the pathname to the socket:

```
struct sockaddr_un addr;
memset(&addr, 0, sizeof(addr));
addr.sun_family = AF_UNIX;
strncpy(addr.sun_path, "socket", sizeof(addr.sun_path)-1);
bind(fd, (struct sockaddr*)&addr, sizeof(addr));
```

The resulting pathname is visible in the filesystem, like:

```
% ls -l
srwxr-xr-x 1 thanson thanson     0 2011-07-02 17:11 socket
```

Unlink before bind

Note that, once created, this socket file will continue to exist, even after the server exits. If the server subsequently restarts, the file prevents re-binding:

```
% ./srv
% ./srv
bind error: Address already in use
```

So, servers should unlink the socket pathname prior to binding it.

File permissions control who can connect

For UNIX domain sockets, file and directory permissions restrict which processes on the host can open the file, and thus communicate with the server. Therefore, UNIX domain sockets provide an advantage over Internet sockets (to which anyone can connect, unless extra authentication logic is implemented).

Comparison with named pipes for IPC

IPC within a Unix host by may be accomplished by several other means including named pipes. What circumstances favor UNIX domain sockets versus pipes? The choice is influenced by these factors:

Duplex
: Stream sockets provide bi-directional communication while named pipes are uni-directional.

Distinct clients
: Clients using sockets each have an independent connection to the server. With named pipes, many clients may write to the pipe, but the server cannot distinguish the clients from each other-- the server has only one descriptor to read from the named pipe. Because the named pipe has only read descriptor and possibly-multiple writers, random interleaving can also occur if a client writes more than `PIPE_BUF` bytes in one operation. Since pipes have these limitations, UNIX domain sockets should be used if there are multiple clients that need to be distinguishable or which write long messages to the server.

Method of creating and opening
: Sockets are created using `socket` and assigned their identity via `bind`. Named pipes are created using `mkfifo`. To connect to a UNIX domain socket the normal `socket`/`connect` calls are used, but a named pipe is written using regular file `open` and `write`. That makes them easier to use from a shell script for example.

Linux Abstract Socket Namespace

Linux has a special feature: if the pathname for a UNIX domain socket begins with a null byte *\0*, its name is not mapped into the filesystem. Thus it won’t collide with other names in the filesystem. Also, when a server closes its UNIX domain listening socket in the abstract namespace, its file is deleted; with regular UNIX domain sockets, the file persists after the server closes it.

Advanced features

Unix domain sockets can pass peer credentials and references to open files between processes. These counter-intuitive abilities utilize ancillary data. See unix(7), that is, `man -s 7 unix` and the subdirectories here for examples.

Resources

Here are some C programs that implement a UNIX domain socket client and server. These are placed in the public domain.

- A server example
- A client example