Unix domain vs internet sockets

> I am coding a daemon program. I am not sure about which type of sockets i should use. Could you compare ip sockets and unix domain sockets? My main criterions are performance and protocol load. What are the differences between impelementations of them at kernel level?

There are a few differences that might be of interest, in addition to the already pointed out difference that if you start out using IP sockets, you don't have to migrate to them later when you want inter-machine connectivity:

– **UNIX domain sockets** use the file system as the address name space. This means you can use UNIX file permissions to control access to communicate with them. I.e., you can limit what other processes can connect to the daemon — maybe one user can, but the web server can't, or the like. With IP sockets, the ability to connect to your daemon is exposed off the current system, so additional steps may have to be taken for security. On the other hand, you get network transparency. With UNIX domain sockets, you can actually retrieve the credential of the process that created the remote socket, and use that for access control also, which can be quite convenient on multi-user systems.

– **IP sockets** over localhost are basically looped back network on-the-wire IP. There is intentionally "no special knowledge" of the fact that the connection is to the same system, so no effort is made to bypass the normal IP stack mechanisms for performance reasons. For example, transmission over TCP will always involve two context switches to get to the remote socket, as you have to switch through the netisr, which occurs following the "loopback" of the packet through the synthetic loopback interface. Likewise, you get all the overhead of ACKs, TCP flow control, encapsulation/decapsulation, etc. Routing will be performed in order to decide if the packets go to the localhost. Large sends will have to be broken down into MTU-size datagrams, which also adds overhead for large writes. It's really TCP, it just goes over a loopback interface by virtue of a special address, or discovering that the address requested is served locally rather than over an ethernet (etc).

– **UNIX domain sockets** have explicit knowledge that they're executing on the same system. They avoid the extra context switch through the netisr, and a sending thread will write the stream or datagrams directly into the receiving socket buffer. No checksums are calculated, no headers are inserted, no routing is performed, etc. Because they have access to the remote socket buffer, they can also directly provide feedback to the sender when it is filling, or more importantly, emptying, rather than having the added overhead of explicit acknowledgement and window changes. The one piece of functionality that UNIX domain sockets don't provide that TCP does is out-of-band data. In practice, this is an issue for almost noone.

In general, the argument for implementing over TCP is that it gives you location independence and immediate portability — you can move the client or the daemon, update an address, and it will "just work". The sockets layer provides a reasonable abstraction of communications services, so it's not hard to write an application so that the connection/binding portion knows about TCP and UNIX domain sockets, and all the rest just uses the socket it's given. So if you're looking for performance locally, I think UNIX domain sockets probably best meet your need. Many people will code to TCP anyway because performance is often less critical, and the network portability benefit is substantial.

Right now, the UNIX domain socket code is covered by a subsystem lock; I have a version that used more fine-grain locking, but have not yet evaluated the performance impact of those changes. I've you're running in an SMP environment with four processors, it could be that those changes might positively impact performance, so if you'd like the patches, let me know. Right now they're on my schedule to start testing, but not on the path for inclusion in FreeBSD 5.4. The primary benefit of greater granularity would be if you had many pairs of threads/processes communicating across processors using UNIX domain sockets, and as a result there was substantial contention on the UNIX domain socket subsystem lock. The patches don't increase the cost of normal send/receive operations, but due add extra mutex operations in the listen/accept/connect/bind paths.

Robert N M Watson
http://lists.freebsd.org/pipermail/freebsd-performance/2005-February/001143.html

—
return to gimbo wiki home page