

Sockets

Sockets are a very common interface that make TCP streams look like file streams. The socket interface shown here has been implemented for many operating systems and languages. This handout shows the Perl versions, but the socket functions look very similar in other languages. A Perl program needs the statement "use Socket;" up near its top to import these functions. In C, the headers are in "sys/socket.h".

Error Handling

When writing Internet code, it's important to do all the error handling, because in a network situation the errors will actually happen with some regularity (host name not found, connection refused...). You need to put in error handling right from the start so you can distinguish the network errors from your own programming errors. Most of the networking functions return false on error. In addition, in the case of an error, they will put a somewhat descriptive string in the global variable \$!.

Buffering -- Autoflush

Many I/O systems do "buffering" automatically for efficiency. When you do a print(), the data is not written immediately. The system buffers up the characters until there is enough data to send all at once. That can be a real problem if the print is part of a protocol where you send, say, one line of text and then wait for a response. In Perl, the FileHandle module provides the "autoflush" function -- sets a file handle so it does not buffer...

```
use FileHandle;

....

## Set the socket to be unbuffered
autoflush SOCK, 1;
```

1. Hostname Lookup

\$ipaddr = inet_aton(\$hostname) -- tries to look up the IP address of a hostname such as "www.yahoo.com". This will probably do a quick transaction with the local DNS server to get the IP address. Fails if the hostname cannot be looked up either because the hostname is wrong, or the local DNS system is down. Returns false on error and sets \$!, since it basically never does.

2. Create Socket Address

\$sockaddr = sockaddr_in(\$port, \$ipaddr); -- given a port number and an ipaddr, combine the two into a socket address that can be passed to connect(). We'll assume this step doesn't fail.

3. Allocate Socket

`socket(SOCK, PF_INET, SOCK_STREAM, 0)` -- given file handle (e.g. `SOCK`, or could be stored in `$sock`), allocate a TCP socket and connect it to the file handle. This just allocates the socket resources; it does not try to connect the socket to anything. The constants shown are to create a stream (virtual circuit) connection using TCP. Other constants can be used to create other sorts of sockets. Fails if there are not enough resources to create another socket on the local machine. Most OS's have some kernel limit on how many simultaneous sockets can be maintained -- probably a few thousand. Returns false on error and sets `$!`.

4. Connect

`connect($sock, $sockaddr)` -- given a socket (from step 3) and a socket address (from step 2), try to connect to the given port on that machine. On success, the given socket can now be used for reading and writing to the remote machine. Reading will block if there is no data available. Fails if the connection cannot be made to the remote machine because it is unreachable or it is not listening on that port number. Returns false on error and sets `$!`.

5 Read

The simplest way to read text data is just with the standard `<SOCK>` file reading operator. Returns `undef` on error or the EOF.

```
connect(SOCK, $sockaddr);
$line = <SOCK>;    ## read a line of text
```

Reading all the data until the EOF is like reading from a file...

```
while (defined($line = <SOCK>)) {
    ...
}
```

Most protocols involve taking turns exchanging a few lines of text back and forth (no EOF) in which case the above while loop will not work. The while loop is only if you want to continually read data until the other end does a `close()`. The function `sysread(SOCK, str, length)` is an alternative that tries to read the given number of bytes, and puts them in the given string. Do not use both `<>` and `sysread()` -- use one or the other.

6 Write

The simplest way to write text is data is with `print`...

```
print SOCK "Hello there, how are you?\012";
```

The function `syswrite(SOCK, str, length)` is an alternative. Do not use both `print` and `syswrite()` -- use one or the other.

7 Close

`close(SOCK)` -- closes the reading and writing capabilities of this end of the socket. The close on the writing end here will show up as an EOF on the other end.

Finger example

```
#!/usr/bin/perl -w
## finger.pl
## A simple example of client-side sockets
##
## The "finger" service listens on port 79
## You send it a user name, it sends back some lines of text
## It used to be that finger worked on most hosts.
## Now however, it's rare, since it allows a bad-guy
## to figure out user names on the host.
## finger.pl nick amy3.stanford.edu works
## finger.pl nick elaine.stanford.edu rejected
## finger.pl president whitehouse.gov -- hangs ?!

use strict "vars";
use Socket;
use FileHandle;

## The end-of-line we'll use
my($EOL);
$EOL = "\015\012";

my($hostname, $user);

(scalar(@ARGV) == 2) || die "usage: <user> <host>";
$user = shift(@ARGV) || die "need username first";
$hostname = shift(@ARGV) || die "need hostname second";

my ($sport, $err, $ipaddr, $sockaddr);

$sport = 79;## port for "finger" service

## Look up the hostname to get its IP addr
$ipaddr = inet_aton($hostname) || die "cannot find '$hostname'";

## Make a socket addr out of the IP+port
$sockaddr = sockaddr_in($sport , $ipaddr);

## Create the socket
socket(SOCK, PF_INET, SOCK_STREAM, 0) || die "cannot create socket";

## Set the socket to be unbuffered
autoflush SOCK, 1;

## Connect the socket
connect(SOCK, $sockaddr) || die "cannot connect";

## Send the username
print SOCK "$user$EOL";
```

```
## Read back and print what they send back until EOF
```

```
my($line);
```

```
while (defined($line = <SOCK>)) {  
    chomp($line);  
    print $line, "\n";  
}
```

```
## Alternately, we could use the perl function  
## sysread(SOCK, str, desired_len) for reading
```

```
close(SOCK);
```

```
elaine26:~/193i> finger.pl nick amy3.stanford.edu  
Login name: nick                      In real life: Nick Parlante  
Directory: /afs/ir/users/n/i/nick      Shell: /bin/tcsh  
No unread mail  
elaine26:~/193i> finger.pl nick elaine26.stanford.edu  
cannot connect at finger.pl line 44.  
elaine26:~/193i> finger.pl president whitehouse.gov      // hangs  
^C
```