

Sockets 2

Part 2 of our coverage of sockets in Perl.

Autoflush

I/O systems may "buffer" write requests together to send them in a larger chunk. This can be a real problem for socket I/O where you often want the bytes sent right away. Here's one way to set a file handle to be unbuffered...

```
use FileHandle;          ## this declaration is required up at the top

...
socket(SOCK, ....);     ## create the handle
autoflush SOCK, 1;      ## set to be unbuffered
```

Here's a more cryptic way to do it...

```
my($temp) = select(SOCK); ## select the file handle
$| = 1; ## set unbuffered
select($temp); ## re-select the old handle
```

Client Example

Here's a simple but complete example of client side sockets.

```
#!/usr/bin/perl -w
## A simple example client-side sockets
##
## The "finger" service listens on port 79
## You send it a string, it sends back the answer

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

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 ($port, $err, $ipaddr, $sockaddr);

$port = 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
$sockaddr = sockaddr_in($port , $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";
}

```

Notes

You may want to separate out your own `CreateClientSocket()` subroutine so you can re-use it. This example uses "die" for its simple error checking -- this is a problem for later programs that don't wish to exit on every error. HW1 should print to `STDERR` and then use `exit(-1)`, so what's printed is cleaner than what die does.

Sample Output

```

elaine0:~/193i> finger.pl nick elaine12.stanford.edu
Login name: nick                               In real life: Nick Parlante
Directory: /afs/ir/users/n/i/nick             Shell: /bin/tcsh
No unread mail
elaine0:~/193i> finger.pl nick no-such-host.stanford.edu
cannot find 'no-such-host.stanford.edu' at finger.pl line 25.
elaine0:~/193i> finger.pl nick www.stanford.edu
cannot connect at finger.pl line 38.
elaine0:~/193i>

```

Server Side Sockets

The server sets up a socket and then waits for an incoming connection. Choose a port # in the range 1024..32767. For server side, the calls `socket` and `sockaddr_in` are as before. The `setsockopt` call can be used to tell the server socket to re-use port #'s for separate connections. The `bind` call associates the server socket with an port #, but does not yet begin listening.

Here's what a nicely decomposed `CreateServerSocket` subroutine might look like...

```

## Given a socket name to use and a port #,
## create a server socket so it's ready for the listen call
## Returns an error string on error, or "" on success.
## usage: $err = CreateServerSocket('SOCK', 80);
sub CreateServerSocket {
    my($sock, $port) = @_;

    socket($sock, PF_INET, SOCK_STREAM, 0) || return("socket $!");

    setsockopt($sock, SOL_SOCKET, SO_REUSEADDR, pack("l",1)) ||
        return("sockopt $!");

    bind($sock, sockaddr_in($port, INADDR_ANY)) || return("bind $!");

    return("");
}

```

Listen

The listen call starts listening on the bound port #. The OS will queue incoming socket connections up to some limit -- maybe 5 or so. The constant, SOMAXCONN uses the system default for the number to queue.

```
listen(SERVER, SOMAXCONN) || die "listen: $!";
```

Accept

Accept blocks waiting for an incoming connection. The second argument should be the server socket. The first argument should be an un-initiated file-handle name which will be set to the incoming client connection. Close the client connection when finished. Returns an address struct identifying the incoming client.

my(\$port, \$iaddr) = inet_ntoa(\$iaddr);
 Unpack the address struct into port and ip addr parts.

my(\$name) = inet_ntoa(\$iaddr);
 Convert the encoded ip addr into a readable string.

Server Example

```

CreateServerSocket(SERVER, $port);
listen(SERVER, SOMAXCONN);

while (1) {
    my $client_addr = accept(CLIENT, SERVER); ## blocks
    my($port, $iaddr) = unpack_sockaddr_in($client_addr);
    my ($name) = inet_ntoa($iaddr);
    print "Connection from $name $port\n";

    ## do things with CLIENT handle

    close(CLIENT);
}

```