15-213: Introduction to Computer Systems

Recitation 12

Monday April 20th, 2009

Outline

- Networking Overview
- Unix Sockets
- Proxylab intro

Networking Overview

- http://ftp.sunet.se/pub/tv+movies/warriors/warriors-700-VBR.mpg

Unix Sockets

- `socket()`
- `bind()`
- `listen()`
- `accept()`
- `connect()`
- `send()`
- `recv()`
- Dealing with errors on Sockets

socket()

```
NAME
       socket - create an endpoint for communication

SYNOPSIS
       int socket(int domain, int type, int protocol);

DESCRIPTION
       socket()  creates  an endpoint for communication
   and returns a descriptor.

RETURN VALUE
       On success, a file descriptor for  the  new  socket
   is  returned.   On error, -1 is returned, and errno is
   set appropriately.
```

To create a standard TCP/IP socket:

```
sock = socket(PF_INET,SOCK_STREAM,IPPROTO_TCP)
```

bind()

```
NAME
       bind - bind a name to a socket

SYNOPSIS
       int bind(int sockfd, const struct sockaddr
  *my_addr, socklen_t addrlen);

DESCRIPTION
       bind()  gives  the socket sockfd the local address
  my_addr.  my_addr is addrlen bytes long.  Traditionally,
  this is called "assigning a name to a socket."  When a
  socket is created with socket(2),  it  exists  in  a
  name space (address family) but has no name assigned.

       It  is  normally  necessary  to assign a local
  address using bind() before a SOCK_STREAM socket may
  receive connections (see accept(2)).
```

bind() Usage (client)

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
addr.sin_addr.s_addr = inet_addr("208.109.109.97");

if((bind(sock,(struct sockaddr *)&addr,
         sizeof(struct sockaddr)))<0){
   perror("Error binding socket");
   /*handle error*/
}
```

bind() Usage (server)

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(port);
addr.sin_addr.s_addr = htonl(INADDR_ANY);

if((bind(sock,(struct sockaddr *)&addr,
        sizeof(struct sockaddr)))<0){
   perror("Error binding server socket");
   /*handle error*/
}
```

listen()

```
NAME
       listen - listen for connections on a socket

SYNOPSIS
       int listen(int sockfd, int backlog);

DESCRIPTION
       To  accept  connections, a socket is first created
  with socket(2), a willingness to accept incoming
  connections and a queue limit for incoming connections
  are specified with listen(), and then the connections
  are accepted  with accept(2).  The listen() call applies
  only to sockets of type SOCK_STREAM or SOCK_SEQPACKET.

      The backlog parameter defines the maximum length the
  queue of pending connections may grow to.

RETURN VALUE
       On success, zero is returned.  On error, -1 is
  returned, and errno is set appropriately.
```

accept()

NAME

```
      accept - accept a connection on a socket
```

SYNOPSIS

```
      int accept(int sockfd, struct sockaddr *addr,
  socklen_t *addrlen);
```

DESCRIPTION

The accept() system call is used with connection-based socket types. It extracts the first connection request on the queue of pending connections, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state.

The argument sockfd is a socket that has been created with socket(2), bound to a local address with bind(2), and is listening for connections after a listen(2).

RETURN VALUE

On success, accept() returns a non-negative integer that is a descriptor for the accepted socket. On error, -1 is returned, and errno is set appropriately.

connect()

```
NAME
          connect - initiate a connection on a socket

SYNOPSIS
     int connect(int sockfd, const struct sockaddr
       *serv_addr, socklen_t addrlen);


DESCRIPTION
          The  connect()  system  call  connects  the socket
  referred to by the file descriptor sockfd to the address
  specified by serv_addr.  The addrlen argument specifies
  the size of serv_addr.  The format of  the  address in
  serv_addr is determined by the address space of the
  socket sockfd; see socket(2) for further details.

RETURN VALUE
          If the connection or binding succeeds, zero is
  returned.  On error, -1 is returned, and errno is set
  appropriately.
```

send()

```
NAME
    send, sendto - send a message on a socket

SYNOPSIS
   ssize_t send(int s, const void *buf, size_t len, int flags);
   ssize_t sendto(int s, const void *buf, size_t len, int flags,
                   const struct sockaddr *to, socklen_t tolen);

DESCRIPTION
        The system calls send(), sendto(), and sendmsg() are
   used to transmit a message to another socket.

        The send() call may be used only when the socket is in
   a connected state (so that the intended recipient is
   known).

RETURN VALUE
        On success, these calls return the number of characters
   sent.  On error, -1 is returned, and errno is set
   appropriately.
```

recv()

```
NAME
       recv, recvfrom - receive a message from a socket

SYNOPSIS
       ssize_t recv(int s, void *buf, size_t len, int flags);

       ssize_t recvfrom(int s, void *buf, size_t len, int flags,
                        struct sockaddr *from, socklen_t *fromlen);

DESCRIPTION
       The  recvfrom()  and  recvmsg() calls are used to receive
   messages from a socket, and may be used to receive data on a socket
   whether or not it is connection-oriented.

       If from is not NULL, and the underlying protocol provides the
   source address, this source address is filled in.  The argument
   fromlen is a  value result  parameter,  initialized  to the size of
   the buffer associated with from.

       The recv() call is normally used only on a connected socket
   (see connect(2)) and is identical to recvfrom() with a NULL from
   parameter.

RETURN VALUE
       These calls return the number of bytes received, or -1 if an
   error occurred.  The return value will be 0 when the peer has
   performed  an  orderly shutdown.
```

Handling Errors

- Establish two “stages” of your code
  - Initialization Stage
    - Do all of your setup here, calling socket, bind, listen..etc
    - Any errors here: Print message and exit!
  - Run Stage
    - Handling connections, reading input, sending output.
    - Any errors here: Print message, but **DO NOT EXIT!**
- Use the `perror()` function
  - Uses `errno` and prints out a reason for the failure.

Handling Errors (cont)

```
if((server_socket = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))<0){
        perror("Unable to initalize server socket!\n");
        exit(ERROR_INIT_SOCKET);
}
```

Fatal failure, EXIT!

```
if(bind(server_socket,serverAddr,sizeof(struct sockaddr)) < 0){
        perror("Unable to bind socket");
        exit(ERROR_INIT_SOCKET);
}
```

Fatal failure, EXIT!

```
if((clientSocket=accept(serverSocket,*clientAddr,&clientLength)) <0){
        perror("Could not acccept client connection! Ignoring,");
        return WARN_NETWORK_ACCEPT;
}
```

Non-fatal failure, continue;

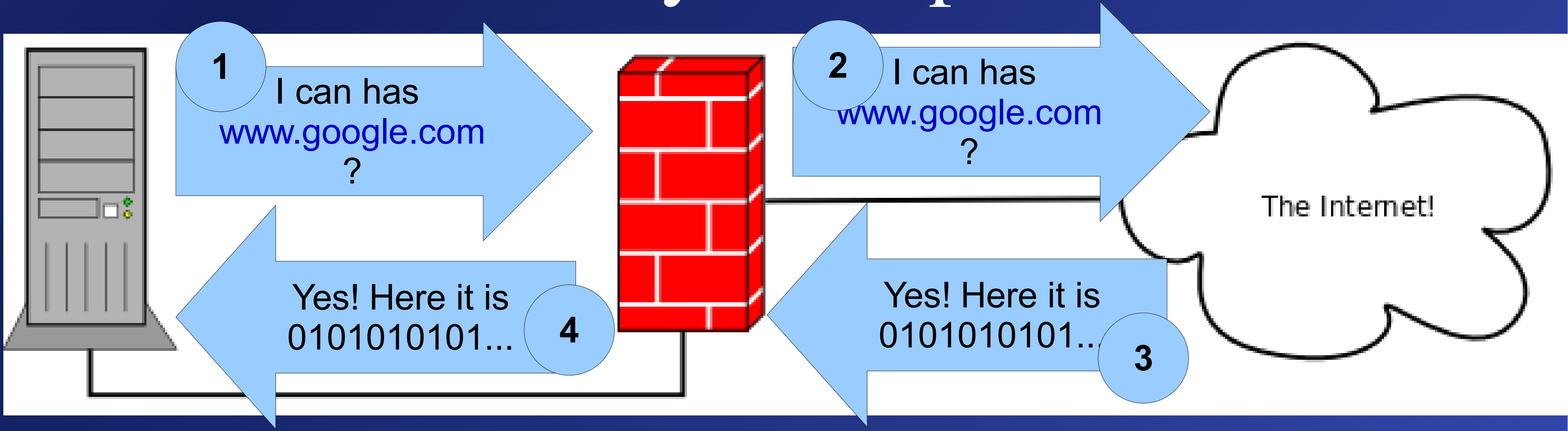
```
if((transmit= send(clientSocket,message,length,0))<0){
        perror("Error Sending!, Ignoring,");
        return WARN_NETWORK_SEND;
}
```

Non-fatal failure, continue;

Intro to Proxylab

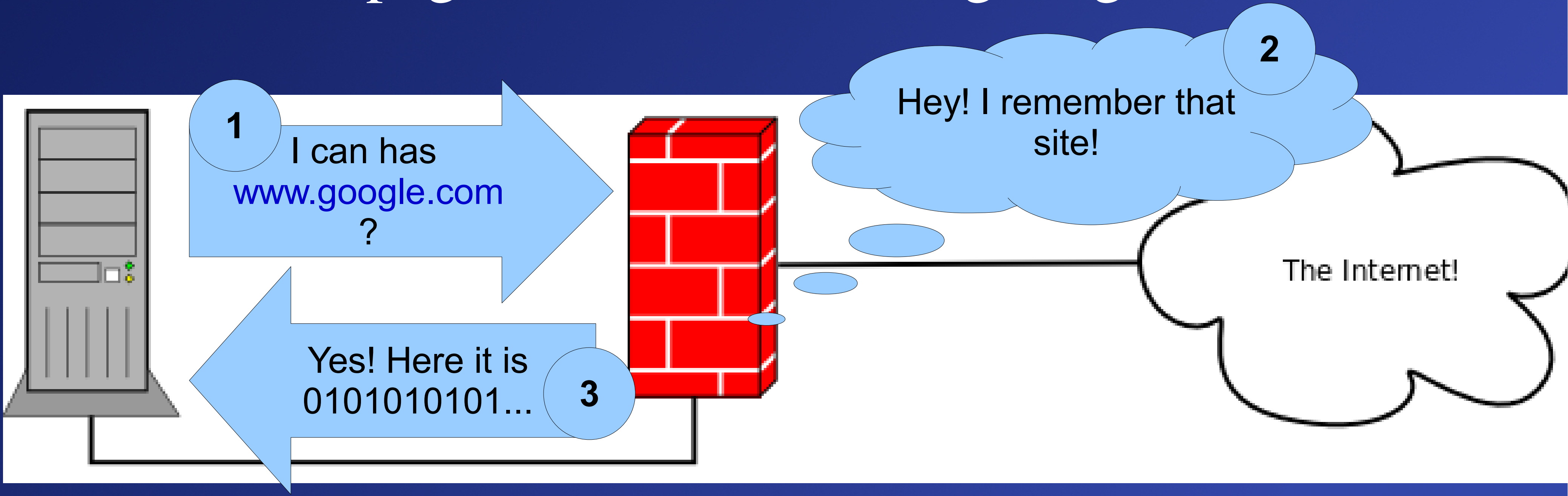
- A Proxy is a server that forwards requests on behalf of users
- Uses:
  - Hide multiple users behind a single IP
  - Provide anonymous access to internet
  - Filter access to internet content
  - 213 projects

Proxy Example!


1
I can has www.google.com ?
2
I can has www.google.com ?
The Internet!
Yes! Here it is 0101010101...
3
Yes! Here it is 0101010101...
4

Caching Proxy

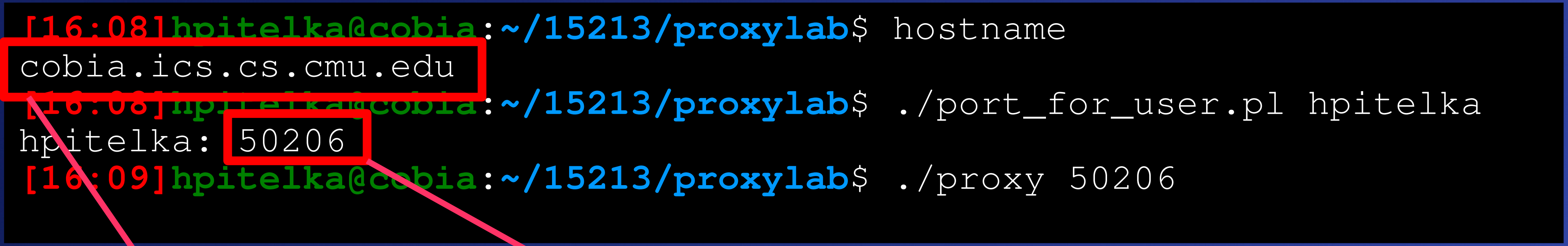
- A proxy that remembers sites it fetches for a user. So if the site is requested again, the proxy server can serve the page, instead of fetching it again.


1
I can has www.google.com ?
2
Hey! I remember that site!
The Internet!
Yes! Here it is 0101010101...
3

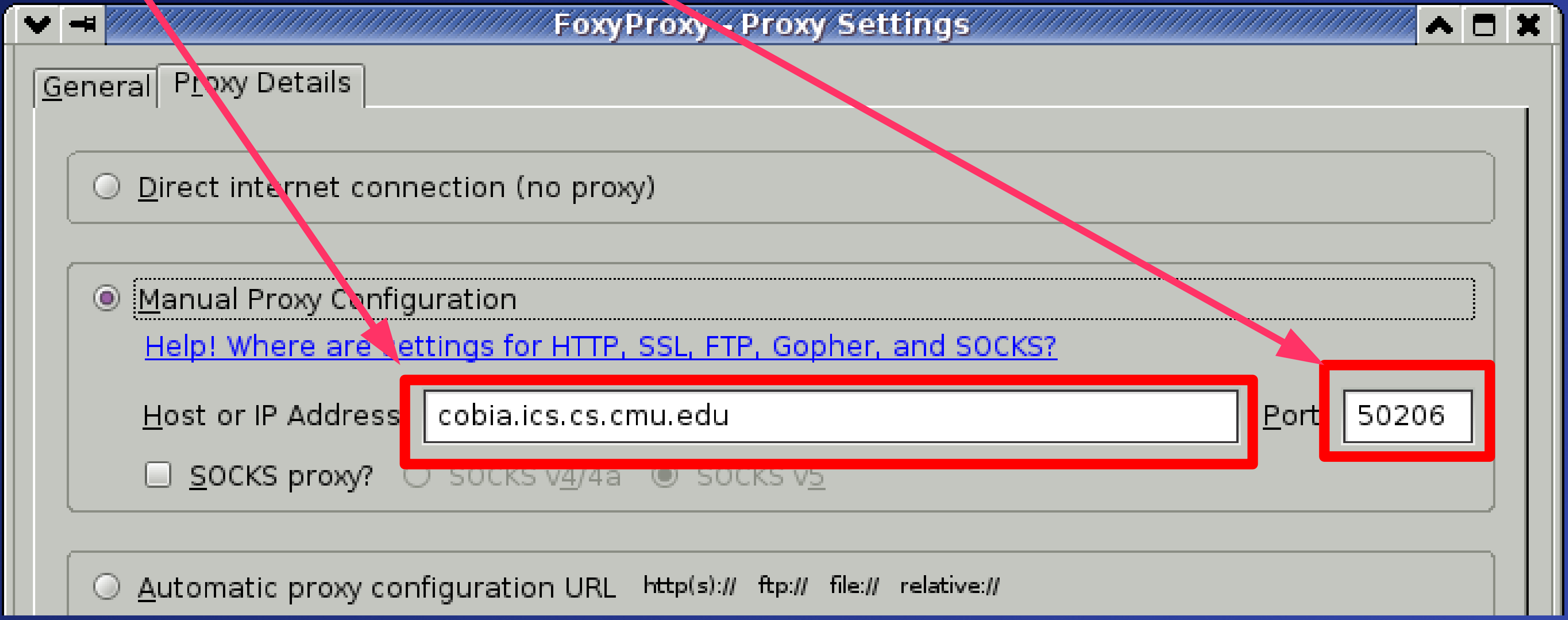
Testing your Proxy

- Zomg no autograder!! :(
- Set up your browser to use your proxy
  - Firefox has a great extension called FoxyProxy:
  - http://foxyproxy.mozdev.org/

Running/Testing your Proxy



```
[16:08]hpitelka@cobia:~/15213/proxylab$ hostname
cobia.ics.cs.cmu.edu
[16:08]hpitelka@cobia:~/15213/proxylab$ ./port_for_user.pl hpitelka
hpitelka: 50206
[16:09]hpitelka@cobia:~/15213/proxylab$ ./proxy 50206
```



FoxyProxy - Proxy Settings

General | Proxy Details

( ) Direct internet connection (no proxy)

(●) Manual Proxy Configuration

Help! Where are settings for HTTP, SSL, FTP, Gopher, and SOCKS?

Host or IP Address: cobia.ics.cs.cmu.edu Port: 50206

[ ] SOCKS proxy? ( ) SOCKS v4/4a (●) SOCKS v5

( ) Automatic proxy configuration URL http(s):// ftp:// file:// relative://

Proxylab Grading

- Grading will be done by a TA by hand after all final submissions.
- We will check for things like
  - proper use of unix sockets
  - no memory leaks
  - no dirty hacks
- So, you need to actually write the assignment, no coding to the traces!

Questions?

kthxbai