Obroni Computer Club – Networking and Web servers

Joachim Breitner

November 7th 2006

It ist time to leave the sixties and go to modern computer times, and look into Networking!

The plan for today is to write a very simple webserver that users can interact from anywhere, using their web browser only. For that we will have a brief look at the HTTP protocol, and how to work with sockets in python.

This is the tenth meeting of the OCC, and there are five more meetings before christmas. After that, I will leave the school, so if you have any special wishes for that time, please tell me.

1 The HTTP Protocol

Whenever you enter a web site, your web browser (Firefox or Internet Explorer or others) contact the web server using the HTTP protocol. The protocol is relatively simple, and text based. That means you can actually understand what goes "on the wire".

To test and explore network services, the tool "netcat" is very useful. Log onto occ and run

```
nc -l -p 10000
```

If you get an error message then the port number (here 10000) might be in use, try any other number larger than 1024. Now make your browser try to access that "server": Enter `http://occ:12345` into the address bar. The browser will just sit there and load. Now look into your console window, and you will see something like:

```
GET / HTTP/1.1
Host: occ:12346
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.7) ↵
    Gecko/20060928 (Debian-1.8.0.7-1) Galeon/2.0.2 (Debian package ↵
    2.0.2-3)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q↵
    =0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: de,en;q=0.5
```

```
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

So this is what the browser tells the server. Most important is what's between GET and HTTP: That is the requested file name!

So now test the other side. Run this command: (80 is the standard web server port)

```
nc occ 80
```

And paste the client code we just gathered, pretending to be a web browser. Add an empty line and the server will respond – in this case with a redirection to the wiki.

Now let's pretend to be a web server again. We run:

```
nc -l -p 10000
```

And after the client sent his request, we type:

```
HTTP/1.1 200 OK
Content-type: text/plain

This is a very simple web site.
```

And we finish with Ctrl-C. The text should apper in the browser.

Now that we have done this by hand, we can create a python program to do so.

2 Simple python webserver

We will do networking the basic way, working directly with sockets. Most of the code will look similar in other programming languages like C, C++, Perl, and should work on most operating systems.

To run a server, four steps are needed:

1. We create a socket. A socket is, like the file object from last session, another kind of variable, this time one to interact with the network. We have to tell the system, what kind of socket we want, in this case an internet socket that transports a stream of data.

2. Then we bind the socket. Think of a real power socket: Binding means wireing it to the cabling. We tell the system on what network and what port we want to receive connections (some computers, like routers, are connected to more than one network).

3. We activate the socket, making it listening for new connections. From now on, clients can connect to the port.

4. Last thing is to actually accept a connection. If there is no connection yet, this will just keep waiting until a connection happens. accept returns, besides the name of

the partner we are talking to, a new socket object for that connection that we can use to read and write to. This step can be reapeated for every new connection, the other three only have to be done once.

To read from a socket, we use the method recv, and we have to pass it the maximum amount of data to read. If we wouldn't, someone could easily crash our server by sending gigabytes of data! If recv returns no bytes it means that the other side has closed the connection. There is no guarantee that recv actually reads all input (the other side might still be sending), but for now we ignore that. To send, we use the method send, or a bit more safe sendall.

The following programs implements that. Change the port number to something unique, and send your webbrowser to that port. Also try reloading the page.

```
1  #!/usr/bin/python
2
3  import socket
4
5  # We create a socket
6  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7  # And wire it correctly to receive connections
8  HOST = ''               # Symbolic name meaning all networks
9  PORT = 10000            # Arbitrary non-privileged port
10 s.bind((HOST, PORT))
11 # Now we activate the socket
12 s.listen(1)
13
14 # Let's do something, in this case, counting requests
15
16 count = 0
17
18 # This will wait for a connection and return a socket and the remote address
19 while 1:
20         conn, addr = s.accept()
21         print 'Connected by', addr
22
23         count += 1
24
25         # For now we hope that all the request is read at once
26         # For a more reliable server, we need to keep recv'ing
27         # until no more data comes
28         request = conn.recv(1024)
29
30         # Strings in three quotes can contain anything
31         # even linebreaks.
32         conn.sendall('''HTTP/1.1 200 OK
```

```
33 Content−type: text/plain
34
35 This is a very simple webserver
36 ''')
37         conn.sendall('And I have seen ' + str(count) + ' requests so far')
38
39         # Lets close the connection
40         conn.close()
```

3 Outputting HTML

Real web pages are not made out of plain text, but they are formatted in HTML. There is no time now to hold a HTML introduction, but if you happen to know HTML or want to learn it yourself (not hard, see http://htmldog.com/), then here is how you output it:

You only need to change what the server is returning. Instead of text/plain you return text/html, and then you return the html code. Example:

```
        conn.sendall('''HTTP/1.1 200 OK
Content−type: text/html

<html>
<head><title>Nicer page</title></head>
<body>This is an HTML page!</body>
</html>''')
```

4 Different pages

A web server usually does not always return the same page. When you enter an address like `http://example.com/some/page.html`, then the web browser will send `/some/page.html` as the requested page. As we want to extract that out of the first line, we can use splitlines, split and join, as seen before:

```
        file = "".join(request.splitlines()[0].split()[1:−1])
```

Using that, let's write a simple web server with several files, whereas the files are saved in a dictionary. If the browser requests no file (that is "/"), we generate an index.

```
1 # same code until s.listen(1)
2
3 # This has the content of the pages
4
5 pages = {
6         '/school': '''<html><head><title>School</title></head>
7                       <body>My school is SOSHGIC</body></html>''',
```

```
        '/country': '''<html><head><title>Country</title></head>
                        <body>My country is Ghana</body></html>''',
        '/food':    '''<html><head><title>Food</title></head>
                        <body>The food quality is doubttful</body></html>''',
}

# This will wait for a connection and return a socket and the remote address
while 1:
        conn, addr = s.accept()

        # For now we hope that all the request is read at once
        # For a more reliable server, we need to keep recv'ing
        # until no more data comes
        request = conn.recv(1024)

        file  = "".join(request.splitlines()[0].split()[1:-1])

        # The header is always the same:
        conn.sendall('''HTTP/1.1 200 OK
Content-type: text/html

''')

        if file == "/":
                conn.sendall('<html><head><title>Index</title></head>'+
                        '<body><ul>')
                for page in pages:
                        conn.sendall('<li><a href="'+page+'"> See '+
                                page+'</a></li>')
                conn.sendall('''</ul></body></html>''')
        else:
                conn.sendall(pages[file])

        # Lets close the connection
        conn.close()
```