CS214 - Advanced UNIX

Lecture 10 - Networking continued. Documents

Ymir Vigfusson

Uses material from

- *The Python Documentation* (http://docs.python.org)

Privileges

- Recall that UNIX has a superuser *root* with UID (user-id) 0.
- Provides privilege separation for:
  - *File system*. Can't access or execute folders or files that you don't own unless granted permission.
  - *Processes*. Can't signal or intercept other processes.

Privileges

- *Device drivers*, including graphics and sound cards.
- *Kernel communication*. Unauthorized for many of commands.
- *Sockets*. Can't open a port below 1024. (leaves 1025-65535).

Wget

`wget` is a wonderful automatic network downloader

- Example: `wget http://www.google.com/` will grab index.html into the current folder.
- `wget -r` tries to recurse directories, useful for producing a local mirror of a site. Use `-nd` to omit creating a local directory structure.
- `wget -c` resumes an earlier download, say if power went out while retrieving a large file.

UNIX networking commands

- `nslookup`, `host` and `dig` are utilities for interacting with DNS servers.
- If you're feeling amibitious, `tcpdump -n` will list all network traffic that is received by the network interface card.

UNIX networking commands

Making your own client and server.

- The `nc` (netcat) utility is handy for connecting and receiving data on arbitrary ports.
  `nc -vlp 5000 -c ./script` will listen on port 5000 for a connection, and execute `script` as a server.
- `netstat -ap` will give you a list of local network connections and open ports, along with each responsible process.

Sockets in bash

- Recent versions of bash allow scripts to connect to arbitrary places using the `/dev/tcp` interface. It's disabled on Debian/Ubuntu per default.

```
exec 3<>/dev/tcp/www.cornell.edu/80
echo "GET / HTTP/1.0" >&3
cat <&3
```

Socket primitives

Initially, low-level communication code can look intimidating.

- Not fundamentally complex things going on.
- For clients we want to get a socket, connect it to a remote host, and communicate.
- For servers, we want to bind a socket to a port, listen for clients, accept them and then communicate.

Socket primitives in Python

- First must `import socket`.
- Get a TCP socket using `s=socket.socket(AF_INET,SOCK_STREAM)`.
- UDP would use `SOCK_DGRAM` instead.
- Local UNIX sockets for inter-process communication would use `AF_UNIX`.

Client code in Python

- Then connect to a server.
- Use `s.connect((HOST,PORT))`. Note use of tuple.
- For example, `s.connect(('www.google.com',80))`.
- Python is nice about resolving DNS names for you and saves you a headache about network byte order (endianness).

Client code in Python

- Data is sent using `s.send('text')`
- Request is received using `buf = s.recv(1024)` (or any arbitrary size).

Client code in Python

- Reading and writing can be made more convenient by transforming the socket into a file.
- `f = s.makefile('rw', 0)` accomplishes this.
- Now you can do
  e.g. `f.write(f.readline().strip())`.
- Remember to close both the file and socket using `f.close()` and `s.close()`.

Client in Python

```
from socket import *
s = socket(AF_INET, SOCK_STREAM)
s.connect((’keegatahw.com’, 5000))
s.send(’Hello, world’)
data = s.recv(1024)
s.close()
print ’Received’, repr(data)
```

Server in Python

- For servers, we still open up a socket `s`
- We then `bind` the socket to a port.
- `s.bind((HOST,PORT))` where host is usually just ” for the current hostname.
- `s.listen(1)` will allow a backlog of at least 1 client (system max. 5) to wait for the port.

Server in Python

- For non-intuitive reasons, we often use

```
c.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEA
```

- This allows ports to be reused even though active TCP connections are still timing out.

Server in Python

- We can now wait for a client by calling `accept`
- `conn,addr = s.accept()`
- Here `conn` is a socket for that particular client.
- `addr` is a structure containing the remote IP address, and some other useful information about the client.

Server in Python

- We can now communicate with `conn` like we did with the `s` socket in the client.
- What if we wish to use non-blocking routines? We use `select` like in C.

Non-blocking sockets

- First use `s.setblocking(0)`

```
ready_to_read, ready_to_write, in_error =
   select.select(potential_readers,
                 potential_writers,
                 potential_errs,
                 timeout)
```

Server in Python

```
from socket import *
s = socket.socket(AF_INET, SOCK_STREAM)
s.bind(('', 5000)) #localhost, port 5000
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

Open Office

How do you write documents?

- Open office (`ooffice`) is an open-source suite based on Sun's commercially developed Star Office.
- Competitive to other WYSIWYG (what you see is what you get) products, including MS Office. Highly portable, but sometimes slow.

LaTeX

How do you write *pretty* documents?

- TeX written by Stanford professor Donald Knuth to typeset his Art of Computer Programming Books beautifully in 1970s.
- LaTeX is still the standard of most math, physics and engineering discples.
- LaTeX is a mark-up language, much like HTML. Documents must be compiled into final output.

Viewing documents

How do I view them?

- Plethoria of PS and PDF viewers.
- `gv` (ghostview) is based on Ghostscript and has a good renderer for both formats).
- `evince` and `xpdf` work too.
- Adobe has released their reader for UNIX.

Printing

What about printing?

- The *line printer daemon* `lpd` is responsible for maintaining a database of printers that you can contact.
- Use `lpr -P printer document` to send a document to a printer.
- Supports various types of servers, from local USB ports to SAMBA to TCP/IP.