Web Server Basics

CS 360 Internet Programming

Daniel Zappala

Brigham Young University
Computer Science Department

Steps in Handling an HTTP Request

1. read and parse the HTTP request message
   - use supplied HTTP parser
2. translate the URI to a file name
   - need web server configuration to determine the document root
3. determine whether the request is authorized
   - check file permissions or other authorization procedure
4. generate and transmit the response
   - error code or file or results of script
   - must be a valid HTTP message with appropriate headers
5. log request and any errors

Handling Multiple Roots

- use the Host header to find the host name
- configuration file gives the root directory for each host served by the web server
- append the URI path to the root directory to get the complete path

Checking File Permissions

- call open() to determine whether you can access the file

```
try:
  open(filename)
except IOError as (errno,strerror):
  if errno == 13:
    // 403 Forbidden
  elif errno == 2:
    // 404 Not Found
  else:
    // 500 Internal Server Error
```

Accessing File Attributes

- use os.stat(filename) to access file size and last modification time
- use in Content-Length and Last-Modified headers

```
size = os.stat(filename).st_size
mod_time = os.stat(filename).st_mtime
```

Handling Pipelined Requests

(threaded server)

- loop forever
  - read from socket until end of message (`\r\n\r\n`)
  - process HTTP message
- break out of loop for
  - recv() error
  - socket closed

Handling Pipelined Requests

(event-driven server)

- read from socket
- append to cache
- check for end of a message (`\r\n\r\n`)
- process HTTP message if present
- leave any remainder in the cache
- close socket and remove cache if recv() error or recv() returns zero bytes

Unresponsive Clients

(threaded server)

- setup a timeout for the socket
- read actual timeout value from configuration file
- recv() will return raise an exception with errno == EAGAIN on timeout

```
seconds = 1
useconds = 0
opt = struct.pack('ll',seconds,useconds)
s.setsockopt(SOL_SOCKET, SO_RCVTIMEO,opt)
```

Unresponsive Clients

(event-driven server)

- use a timeout with poll()
- if the timeout occurs, then close all client sockets and remove them from the polling object

Getting the Time

```
t = time()
```

- returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds, as a floating point number

Converting to GMT

```
gmt = time.gmtime(t)
```

- takes as input the tieme in seconds since the epoch
- returns a structure that uses GMT

Converting to RFC 822, 1123 Time Format

- the recommended date format for HTTP
- used in the Date and Last-Modified headers

```
format = ''%a, %d %b %Y %H:%M:%S GMT''
time_string= time.strftime(format,gmt)
```

- takes a format string, GMT time struct
- returns a string using RFC 1123 format
- see `http://docs.python.org/library/time.html`

From Time to Time

```
def get_time():
    gmt = time.gmtime(t)
    format = ''%a, %d %b %Y %H:%M:%S GMT''
    time_string= time.strftime(format,gmt)
    return time_string

t = time()
current_time = get_time(t)
mt = os.stat(filename).st_mtime
mod_time = get_time(mt)
```