1

The Internet

- Fundamental Concepts
- Network Programming
- Security Concerns
- Session Handling
- Advanced Network Programming

2

The Hypertext Transfer Protocol

- A Typical HTTP-Session
- Requests and Responses
- Content Negotiation
- Access Control/Password-Protected Pages
- Caching (Proxies)
- State Management
- Authorization

Advanced Network Programming

socketserver – A framework for network server

- Network Services
  - applications available on a communications network
  - example: bulletin board

- can be written in two ways:
  - using the Python socket module – which provides the BSD socket interface
  - using the `socketserver` module which provides specialized framework for writing network servers

I.1

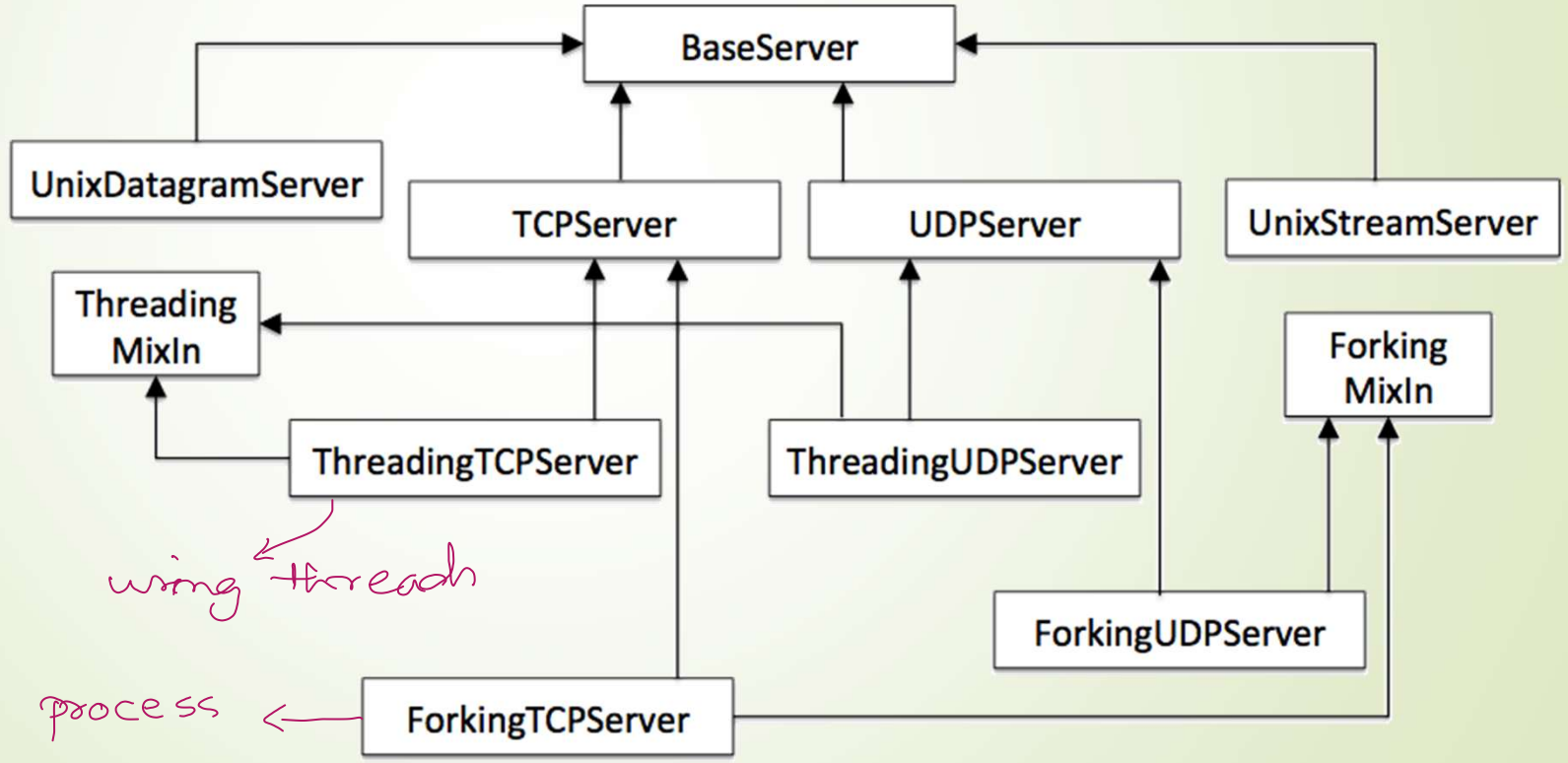
Advanced Network Programming

socketserver – A framework for network server

- set of questions to be answered
  - what kind of a network service it is?
  - should the network service support a protocol that is connection oriented, reliable, and supports Sessions?
  - or the network service support should support a protocol, which is discrete, datagram based, unreliable but still OK for the task in hand?
  - will the Client connections be processed one by one?
  - will the clients making connections to the network service wait patiently while requests are processed one by one?
  - or is it necessary to handle multiple requests concurrently?
  - if Multiple Requests are to be handled concurrently - will separate threads be used for handling each request - or separate processes be used to process each thread?

Advanced Network Programming

socketserver – A framework for network server



Advanced Network Programming

socketserver – A framework for network server

- `class socketserver.BaseServer(server_address, RequestHandlerClass)`
  - superclass of all server objects
  - defines an interface, but does not implement most of the methods
  - two parameters are stored:
    - `server_address`
    - `RequestHandlerClass` attributes

Advanced Network Programming

socketserver – A framework for network server

- `class socketserver.TCPServer(server_address, RequestHandlerClass, bind_and_activate=True)`
- `class socketserver.UDPServer(server_address, RequestHandlerClass, bind_and_activate=True)`
  - uses the TCP resp. UDP protocol
  - if `bind_and_activate` is true, the constructor automatically attempts to invoke `server_bind()` and `server_activate()`

Advanced Network Programming

socketserver – A framework for network server

- classes process requests synchronously
  each request must be completed before the next request can be started
- not suitable if each request takes a long time to complete
  - create a separate process or thread to handle each request
  - `ForkingTCPServer` and `ThreadingTCPServer` classes can be used to support asynchronous behaviour

Advanced Network Programming

socketserver – A framework for network server

- steps
  - create a request handler class by subclassing the `BaseRequestHandler` class and overriding its `handle()` method - this method will process incoming requests
  - instantiate one of the server classes, passing it the server's address and the request handler class
  - call the `handle_request()` or `serve_forever()` method of the server object to process one or many requests
  - finally, call `server_close()` to close the socket (unless you used a with statement)

Advanced Network Programming

socketserver – A framework for network server

- `class socketserver.BaseRequestHandler`

  superclass of all request handler objects, defines the interface

  - `setup()`
    called before the handle() method to perform any initialization actions required
  - `handle()`
    does all the work required to service a request
    - the request is available as `self.request`
    - the client address as `self.client_address`
    - and the server instance as `self.server`
  - `finish()`

Advanced Network Programming

socketserver – A framework for network server

- `handle_request()`
  process a single request. This function calls
  - `get_request()`
    - accept a request from the socket
    - return a 2-tuple containing the new socket object to be used to communicate with the client, and the client's address
  - `verify_request()`
  - `process_request()`
    - calls `finish_request()` to create an instance of the `RequestHandlerClass`
    - if desired, this function can create a new process or thread to handle the request