CMPT 371

Data Communications and Networking

Socket Programming

The socket interface

- The socket interface is used in application programs to specify the interface between the application program and the transport layer protocol software (TCP, UDP)
- The form of the interface (API) may vary for different OS's and between different computer languages.
- The socket interface discussed here is the Python interface which is based on the BSD UNIX sockets
- The Python interface is similar to the BSD UNIX C interface but easier to use.

IO: FILE vs. NETWORK

- When reading or writing a file a file descriptor is used to specify the file referred to
  - A file is opened using a particular file descriptor, binding the file or device to that file descriptor
  - Data is read from and/or written to file
  - to read or write need at least three pieces of information
    1. File descriptor
    2. Buffer (holds data read from file or data being written to the file)
    3. Number of bytes to read/write (or size of buffer)
  - File is closed

NETWORK IO

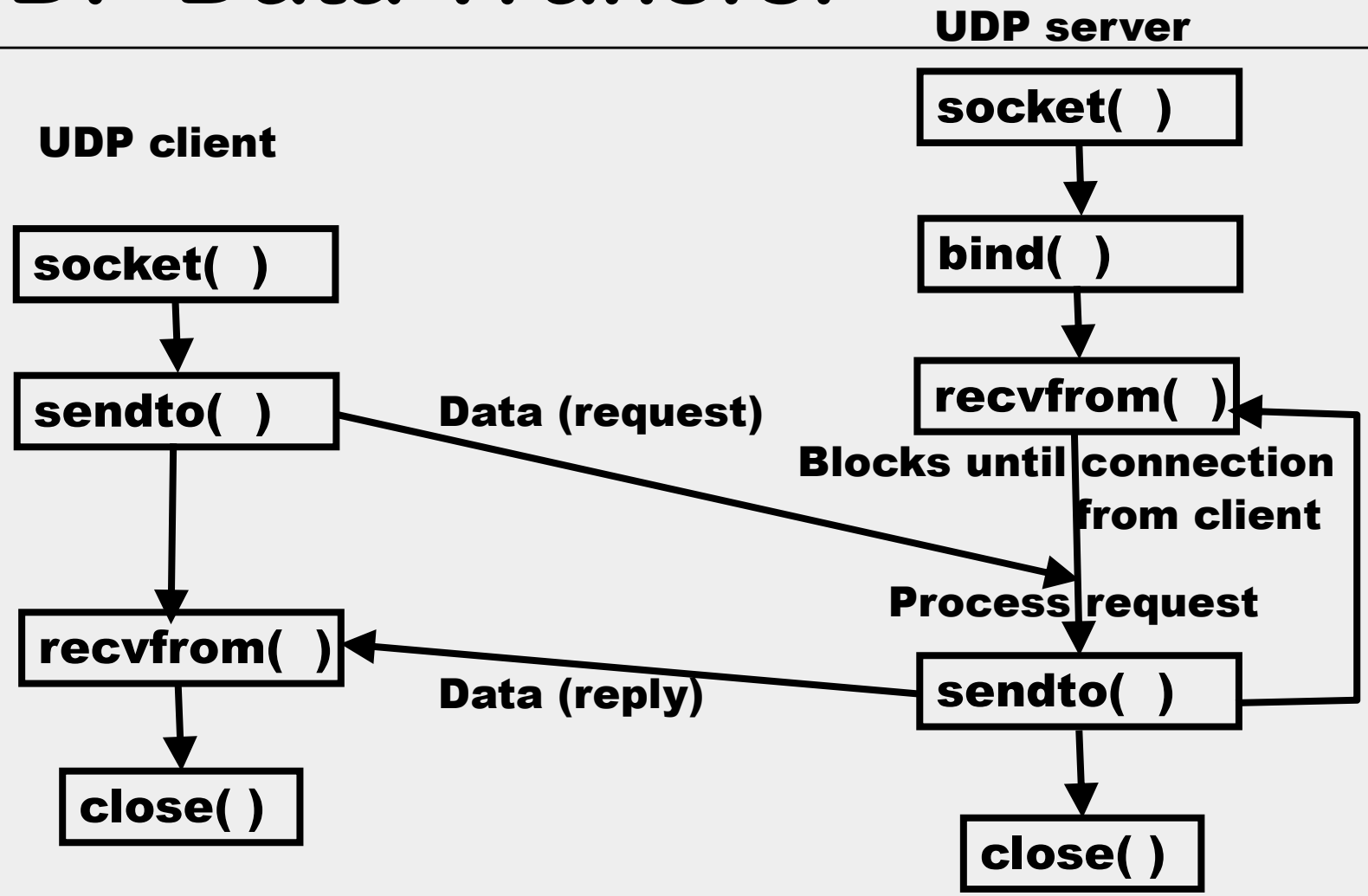
- A socket is a communication endpoint, the socket descriptor is an integer describing a particular socket
- A socket is not automatically bound to a particular IP or port
- The application programmer can choose when to bind the socket to a IP and port, or to let the OS bind to a IP and port at runtime

- Two sockets can be connected to form a connection
- When a connection is made it is assigned a connection identifier
- When sending or receiving data across a network connection we write or read to a connection identifier

IO: FILE vs. NETWORK

- When reading or writing through a socket a connection descriptor is used to specify the particular connection
  - A socket is connected to another socket (communication endpoint) to create a connection
  - A created connection has a connection identifier
  - Data can then be transferred from one endpoint to the other. Once the sockets are connected the connection descriptor can be used like a file descriptor
  - When the data transfer is complete the connection is closed

UDP Data Transfer



UDP client

```
from socket import *
serverName = 'localhost'
serverPort = 50007
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input('Input lowercase sentence:')
clientSocket.sendto( message.encode(),
        (serverName, serverPort) )
modifiedMessage, serverAddress =
        clientSocket.recvfrom(2048)
print (modifiedMessage.decode())
clientSocket.close()
```

Import and localhost

from socket import *

- Imports the python socket library into your code
  - The python library is built on the BSD socket library

serverName = 'localhost'

- This example talks between a server and client on the same machine.
  - The name of the host that is used to connect to other processes on the same host is localhost (or lo)
  - To connect to another host this would be the DNS FQDM name of the host

Storing port and address

serverName = 'localhost'
serverPort = 50007

- This set values for two variables that will be reused by the udp client
- Note in this example I use a different port for the server
  - 1200 (text used in the example) is the “well know” port for UDP. Your system UDP server will use this port.
  - I am using an unassigned port that will not interact with any UDP servers already running on your machine

Create and send a message

clientSocket = socket(AF_INET, SOCK_DGRAM)

- First need to create a best effort socket
  - SOCK_DGRAM tells the client to create a UDP socket
  - AF_INET tells the client to create and IPv4 socket
  - The name the socket is referred to by the rest of the statements in the client will be clientSocket

The socket method:

socket(AF_INET, SOCK_STREAM, PROTOCOL)

- The family indicates the family of protocols that will use the socket (AF_INET for IPv4, AF_INET6 for IPv6)
- The type indicates the particular type of transfer that will be used (SOCK_STREAM for TCP, SOCK_DGRAM for UDP, SOCK_RAW for raw data)
- The protocol indicates the particular protocol to use (usually 0) This argument is optional (defaults to 0)

Creating a socket

socketfd = socket(AF_INET, SOCK_STREAM);

- socket is the socket descriptor for the newly created IPv4 TCP socket
- This socket is not yet associated with any communication endpoint (IP address, port pair)
- The socket is an active socket (for use in active connection mode)
- You can see all the possible values for protocol family, protocol type and protocol in the descriptions of the python interface available online

Create and send a message

```
message = input('Input lowercase sentence:')
clientSocket.sendto( message.encode(), (serverName, serverPort) )
```

- First create the message called message
- Send message through socket clientSocket
  - The message will be sent to the Ip address that corresponds to the DNS FQDN in variable serverName.
  - The ip address will be found using a DNS query made by the sendto method
  - When the message arrives at the destination ip address it will be sent to the process at port serverPort

Receiving a reply

modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)

- When the server receives the message the message will be processed and will produce a reply
- The client needs to receive the reply
  - Here modifiedMessage is the reply
  - The reply arrives from the host with ip address serverAddress
  - The reply is received by clientSocket

Close your socket

- After all information has been transferred close your client's socket
- In our example the name of the client's socket is clientSocket, so to close the socket

```
clientSocket.close()
```

UDP server (Python)

```
from socket import *
serverHost = "127.0.0.1"
serverPort = 50007
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(( serverHost, serverPort))
print ( 'The server is ready to receive ' )
while True:
        sentence, clientAddress = serverSocket.recvfrom(1024)
        print ("Message received: " + sentence )
        capitalizedSentence = sentence.decode().upper()
        serverSocket.sendto(capitalizedSentence.encode(), clientAddress)
```

Setting address of server

serverHost = "127.0.0.1"

serverPort = 50007

- Notice that the serverPort is the same in the client and the server. This is necessary for the connection to be made. Both variables refer to the name of the port on the server.
- 127.0.0.1 is the ip address of the localhost. In LINUX we could also say ‘ ‘ (not in Windows)

Binding to an address (1)

- serverSocket.bind((serverHost, serverPort))
- A server needs to have a specified address / port so that the client can find it
  - For many servers this will be the well know port for the protocol being implemented
    - Particular ports are assigned to particular protocol X these ports are called the well know ports for protocol X
  - To specify a particular port and/or ip address is to be used on a particular socket that port and/or address must be bound to the socket

Binding to an address (2)

- serverSocket.bind((serverHost, serverPort))
- A server needs to have a specified address / port so that the client can find it
- A client can use any port, when it asks to connect to the server it will provide the information on it's own ip address and port
  - The client did not bind to an address or port
- This way many different clients with different ip addresses and/or ports can use the same server

The bind function

- The bind function associates the socket descriptor with a local (same host that the process doing the bind is running on) communication endpoint

  serverSocket.bind(serverName, serverPort)

- ‘ ‘ instead of serverName means all available interfaces (LINUX), this means the server will accept requests regardless of which of its interfaces the requests arrive on

Waiting for a connection

- The listen function converts an unconnected socket into a passive socket.
  - A passive socket is able to accept a connection
  - When created using socket the socket is an active socket (can request a connection but not accept one)
- Listen is usually used by a server process.
- The OS kernel queues requests for connection to the passive socket.

The listen function

socketfd.listen(backlog)

- The backlog indicates the maximum number of connections the kernel (OS) should queue for this socket.

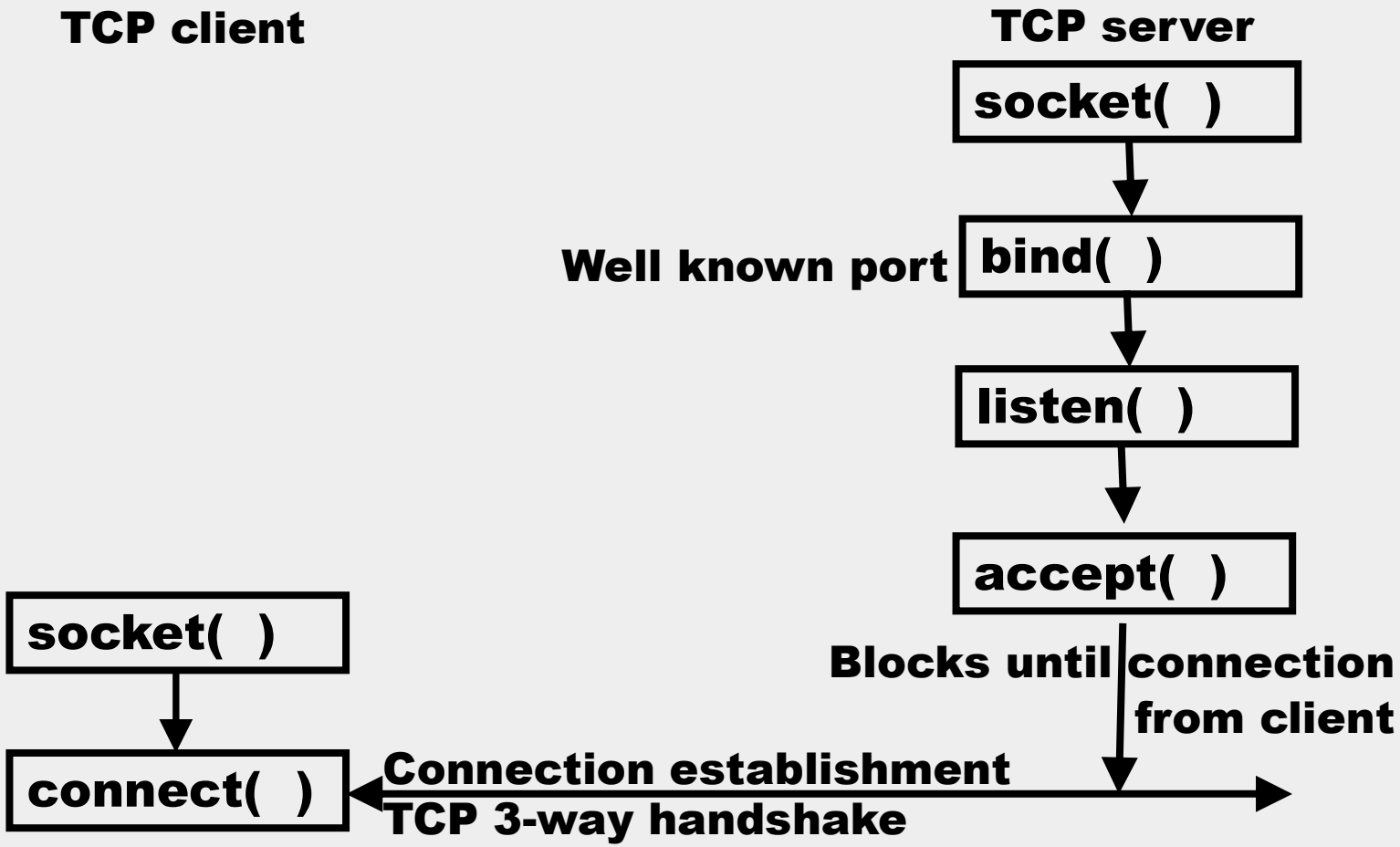
Close your socket

- After all information has been transferred close your servers's socket
- In our example the name of the server's socket is serverSocket, so to close the socket

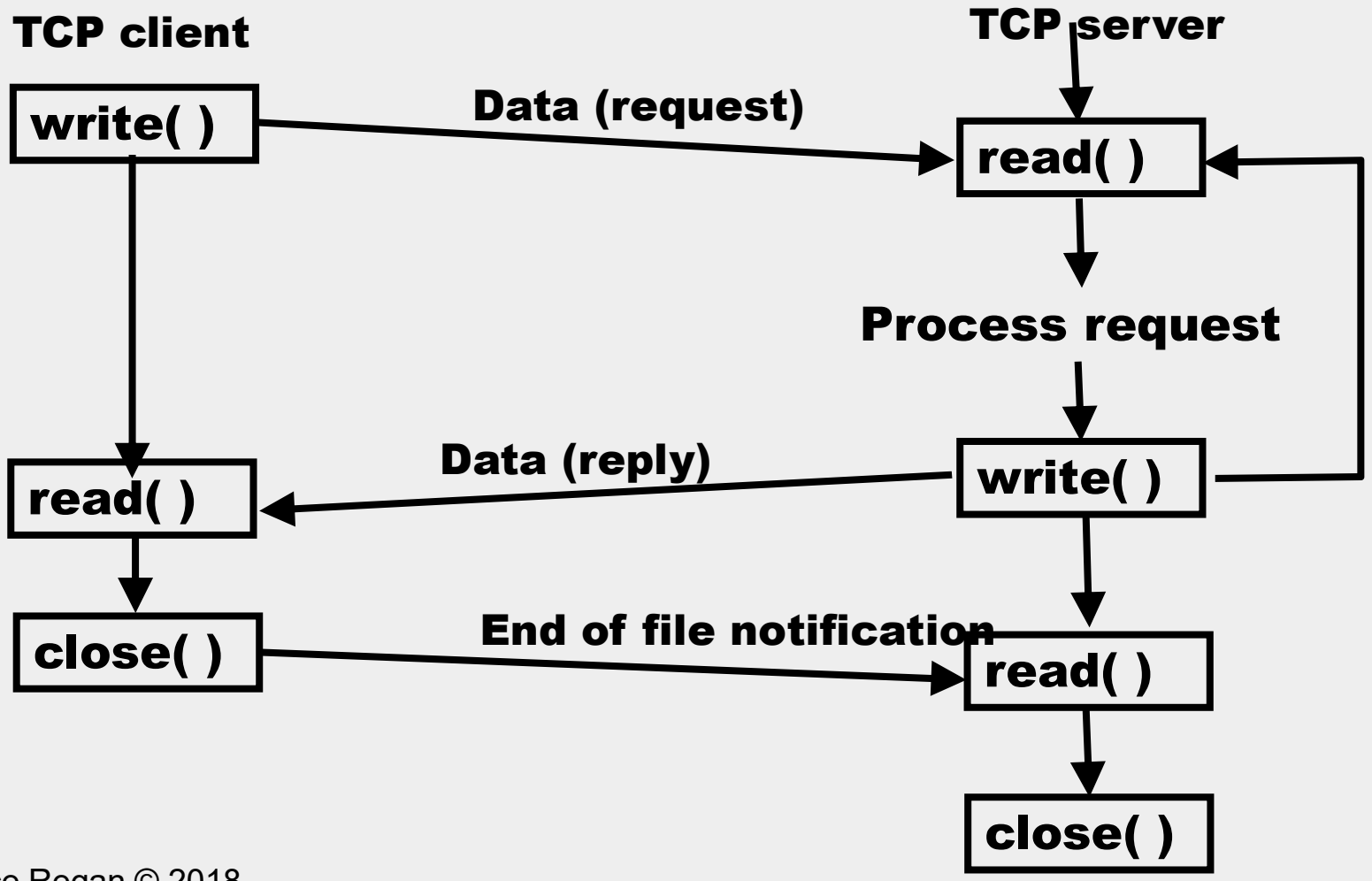
```
serverSocket.close()
```

- The socket is not closed in the example. Why?

Establish connection



TCP Data Transfer:



TCP server 1 (Python)

```
for socket import *
serverPort = 50007
serverHost = 'localhost'
serverSocket = socket.socket(AF_INET, SOCK_STREAM)
serverSocket.bind((serverHost, serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
        :
```

TCP server 2 (Python)

```
while True;
        connectionID, addr = serverSocket.accept()
        sentence = connetionID.recv(2038)
         capitalizedSentence = sentence.upper()
        connectionID.send(capitalizedSentence)
                :
```

Close your socket

- After all information has been transferred close your servers's socket
- In our example the name of the server's socket is serverSocket, so to close the socket

```
serverSocket.close()
```

- The socket is not closed in the example. Why?

local communication endpoints

- The local port and IP address of the host's interface can be determined automatically within the sockets software at run time. If a port or IP address is not specified then:
  - An interface to the network will be specified for you (local IP address)
  - An available (ephemeral) port on the host will be chosen to associate with the endpoint
- Normally servers bind to a specified port, clients allow the OS kernel to choose an interface (the one that can connect to the server) and an ephemeral port

Specifying Destination Address

- In the connect function associates the socket descriptor with a destination address [(IP address, port) port pair specifying the destination connection endpoint]
  - For TCP this function initiates the three way handshake.
  - A client normally requests a connection using the connect function. A server normally waits for that connect request.
- A socket can be connected regardless of whether it has been bound to a local address. If no call has been made to bind the OS kernel will assign the local ephemeral port and select a local IP address

The connect function

socketfd.connect(servername, serverport)

- The socket address specifies the IP address and port number of the destination connection endpoint

Closing a connection

- The close functions default results are
  - mark the socket with socket descriptor sockfd as closed
  - For a TCP connection, start the three way handshake to terminate the connection by sending a FIN
  - Prevents further use of the socket by the application
  - Allows any previously queued data to be sent before closing the connection

socket.close( )

The socket interface

- We have discussed the basics of the socket interface when dealing with a simple iterative server and clients
- The server will queue connect requests from different clients and deal with them one by one
  - For short requests like sending a single packet this can work well
  - For longer requests that take significant processing time this is not efficient, we would like the server to be able to deal with the requests simultaneously
- The solution is to use a concurrent server, that makes a copy of itself or child to deal with each client