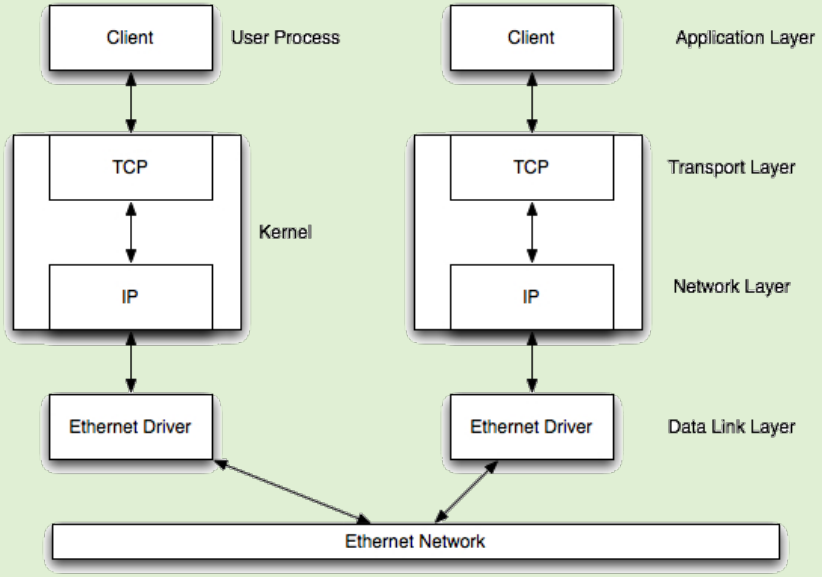
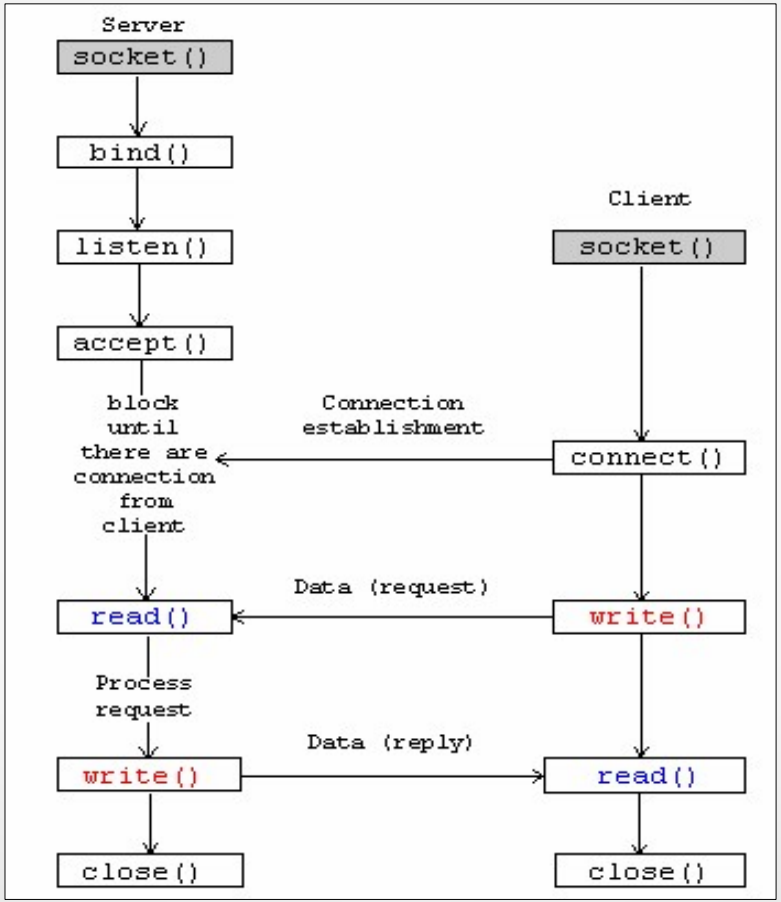
Taller de Programación en Redes
Stack TCP/IP - Sockets



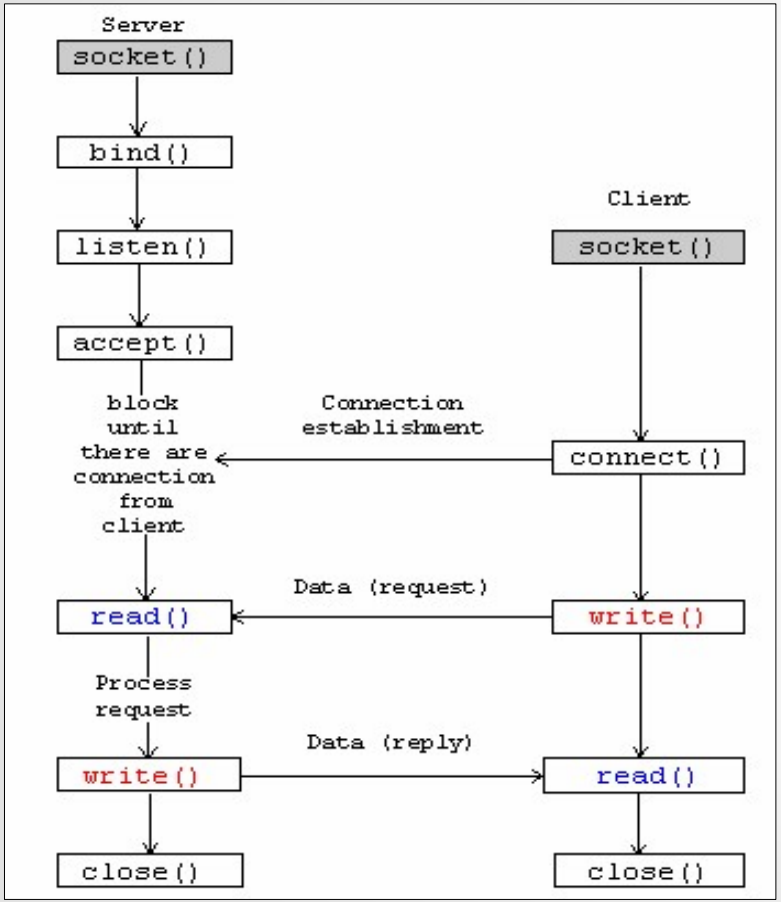
Lic. en Sistemas de Información - Universidad Nacional de Luján
Dr. Gabriel Tolosa – tolosoft@unlu.edu.ar
Lic. Marcelo Fernández – fernandezm@unlu.edu.ar
Clase 2 - Febrero 2018

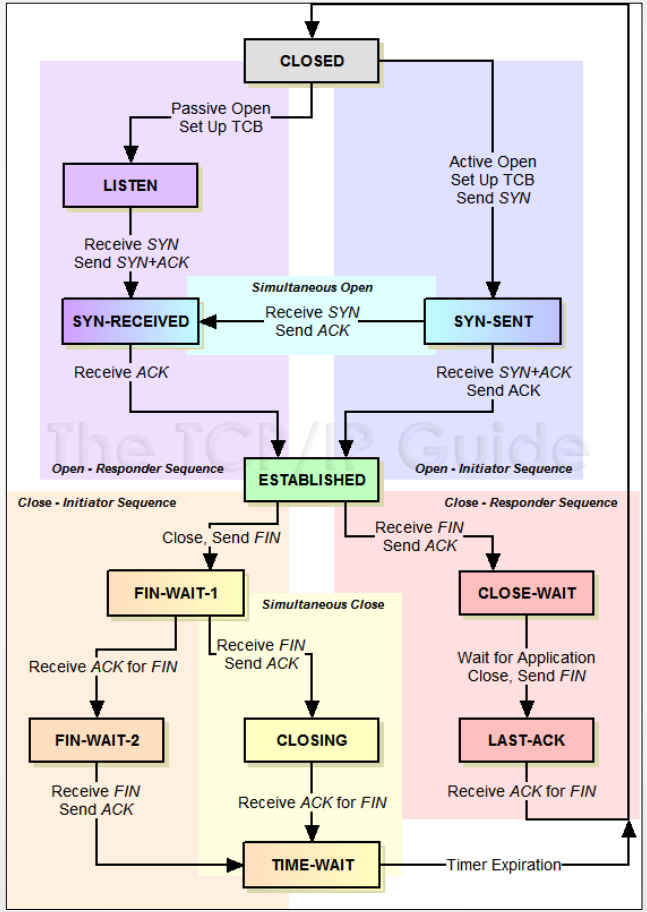
Socket API para TCP– Esquema de Trabajo

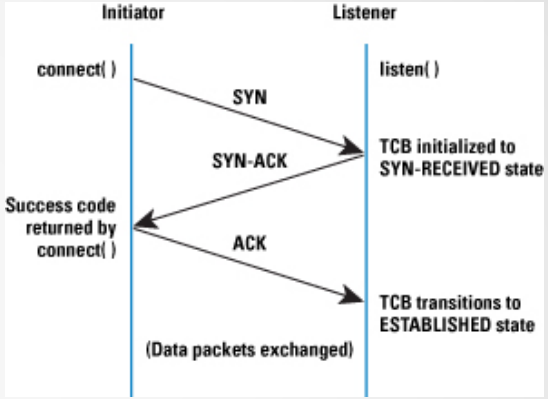


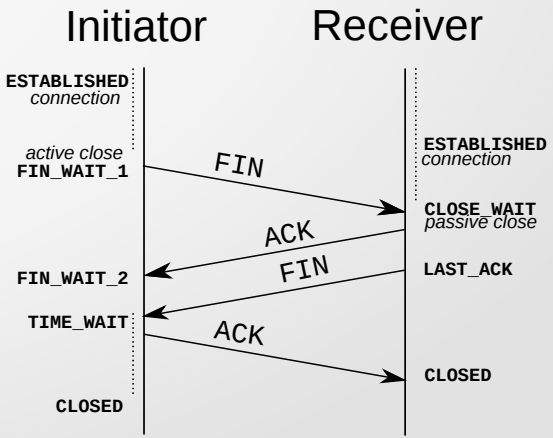
Socket API para TCP– Máquina de Estados

Fuentes
- http://www.tcpipguide.com
- Wikipedia
- Cisco







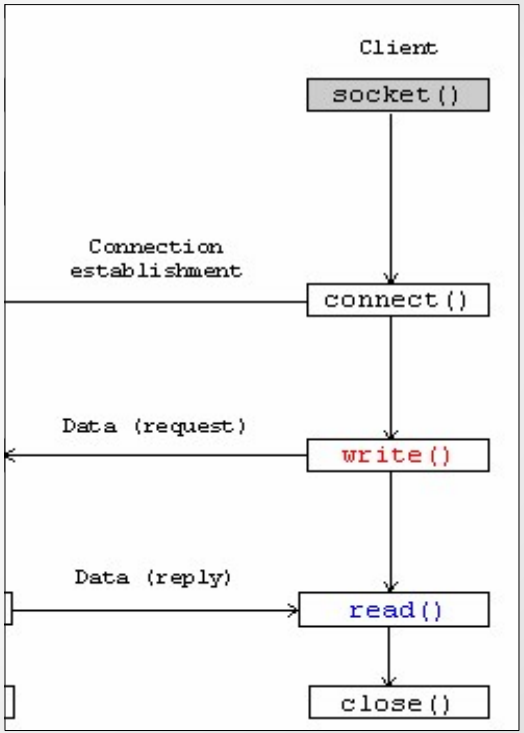


TCP Echo Client

```
# Echo client program
import socket

HOST = '192.168.1.101'  # IP del servidor
PORT = 3500             # Puerto del servidor

# Se crea un socket de tipo Internet (AF_INET) sobre TCP (SOCK_STREAM)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Nos conectamos al host y puerto del servidor
s.connect((HOST, PORT))
# Enviamos un string
s.sendall('Hola Mundo!')
# Leemos la respuesta del socket y lo cargamos en la variable 'data'
data = s.recv(1024)
# Cerramos el socket contra el server
s.close()
print 'Received', repr(data)
```

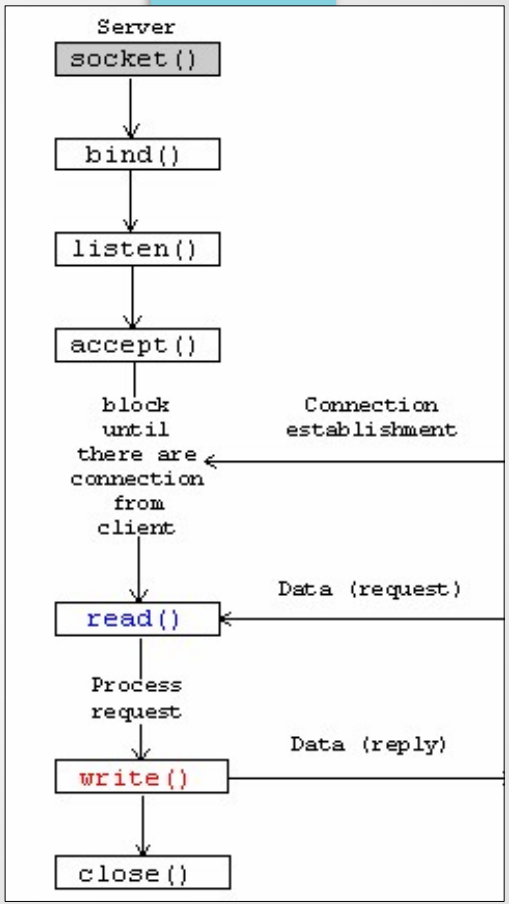


TCP Echo Server

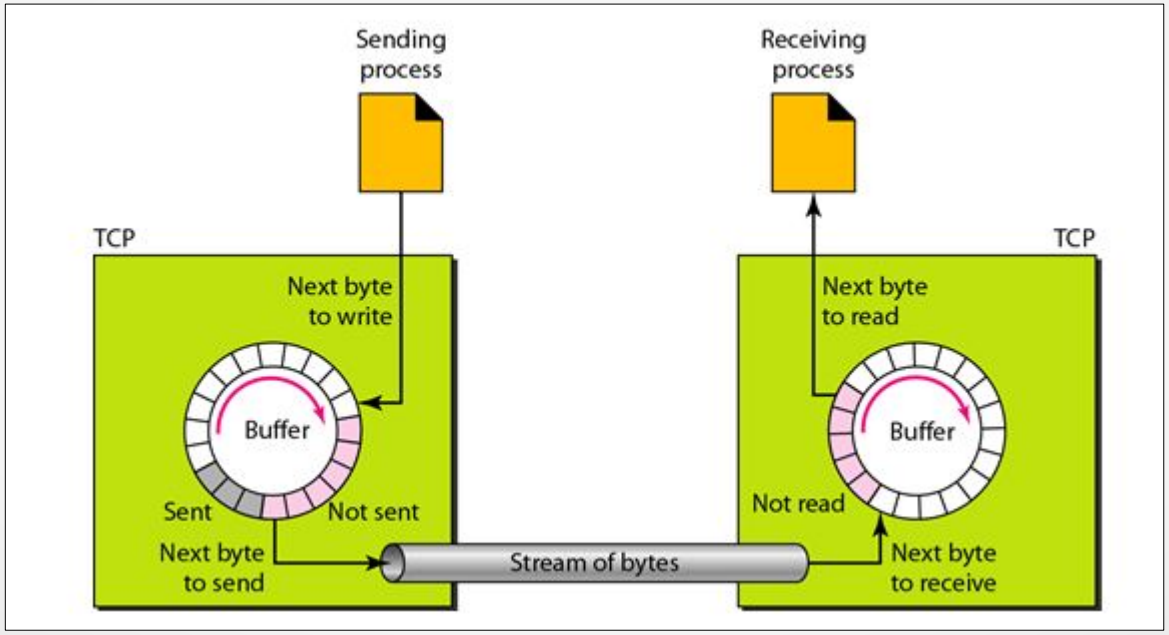
```python
#!/usr/bin/env python
import socket

HOST = '192.168.1.101'    # IP o Hostname donde escucha
PORT = 3500# Puerto de escucha
MAXPEDIDOS = 5     # Cantidad de pedidos acumulados (backlog)
bytes = 1024       # Cantidad maxima de bytes a aceptar

# Se crea un socket de tipo Internet (AF_INET) sobre TCP (SOCK_STREAM)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Seteo opcion REUSEADDR del sol_socket
# Ver https://hea-www.harvard.edu/~fine/Tech/addrinuse.html
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((HOST,PORT))    # Indicamos al socket la direccion IP y puerto
s.listen(MAXPEDIDOS)   # Indicamos el nro max de pedidos que aceptara
while True:
    client, address = s.accept()  # Acepta las conexiones
    # Se obtienen los datos desde el socket cliente
    data = client.recv(bytes)
    print "Conexion desde: %s:%i" %(address[0],address[1])
    if data:
        client.send(data) # Enviamos el echo al cliente
    client.close()        # Cerramos el sock cliente
```



Socket API para TCP – Funcionamiento



TCP - *Stream* de información

Socket API para TCP – Funcionamiento (2)

- No existe socket “servidor” y “cliente”. El rol depende del protocolo.
- Funciones `send(`*`string`*`)`/`recv(`*`bufsize`*`)` (¿Y UDP cómo era?¿Por qué?)
- Manejan buffer de red asociado, ya que es un ***stream***.
- Chequear que se haya enviado/recibido todo lo necesario.
- Si devuelven 0 bytes es porque el socket se está cerrando*
- Ahora bien, los “mensajes” deben:
  - Ser de longitud fija, o bien
  - Estar delimitados por algún *token*, o bien
  - Señalar qué tan largo es, o qué estructura tiene, o bien
  - Usar un socket por cada mensaje o interacción (¿les suena conocido?)

Socket API para TCP – Ejemplo

```
class MySocket:

  MSGLEN = 1024

  def __init__(self, sock=None):
    if sock is None:
        self.sock = socket.socket(
            socket.AF_INET, socket.SOCK_STREAM)
    else:
        self.sock = sock

  def connect(self, host, port):
      self.sock.connect((host, port))

  def mysend(self, msg):
    totalsent = 0
    while totalsent < MiSocket.MSGLEN:
      sent = self.sock.send(msg[totalsent:])
      if sent == 0:
        raise Exception("socket closed")
      totalsent = totalsent + sent
```

Longitud Fija del mensaje

```
def myreceive(self):
  chunks = []
  bytes_recd = 0
  while bytes_recd < MySocket.MSGLEN:
    chunk = self.sock.recv(min(MySocket.MSGLEN-bytes_recd, 2048))
    if chunk == '':
      raise Exception("socket closed")
    chunks.append(chunk)
    bytes_recd = bytes_recd + len(chunk)
  return ''.join(chunks)
```

* Tomado del “Socket Programming Howto”: https://docs.python.org/2.7/howto/sockets.html

Socket API para TCP – Ejercicios Propuestos

1. Realizar la versión “Chat Cliente/Servidor” pero con TCP, usando mensajes de tamaño fijo, reutilizando la clase del ejemplo anterior.
2. ¿Cómo sería la versión con tokens para delimitar los mensajes, siendo el token un espacio o un punto?
3. Implementar una versión con una estructura asociada al mensaje (metadatos), como ser:
   - Longitud
   - ID
   - Timestamp
   - CRC32*

Socket API para TCP – Ejercicios Propuestos (2)

5. Escriba un programa que implemente un cliente HTTP utilizando el módulo socket (similar a wget). Agregue opciones para que el programa use un proxy y guarde en un archivo de log todos los header HTTP recibidos. Además, considere evaluar el header de redirección.

6. Escriba un programa cliente y uno servidor que permitan ejecutar comandos Unix en una máquina remota (por ejemplo: pwd, ls, mv). Ejemplo:

```
#~$ python cliente.py
>>> pwd
>>> /home/usuario/
```

Sus programas debe tener una instancia de autenticación en la cual el nombre de usuario y la clave se encripten antes de enviarse por la red. ¿Cómo se puede resolver este problema? Revise cómo lo hace ssh. Incluya una opción para almacenar en un archivo indicado el log de toda la sesión.

7. Desarrolle un servidor HTTP básico que implemente las respuestas para los códigos HTTP de estado 200 y 404. El servidor retornará el código de estado correspondiente de acuerdo a si existe o no el objeto solicitado. Si el código es 404, debe retornar una página HTML pre-diseñada que contiene el mensaje explicando la situación. Realice pruebas con diferentes navegadores.