# Práctica 8: Sockets

75.59 - Técnicas de Programación Concurrente I

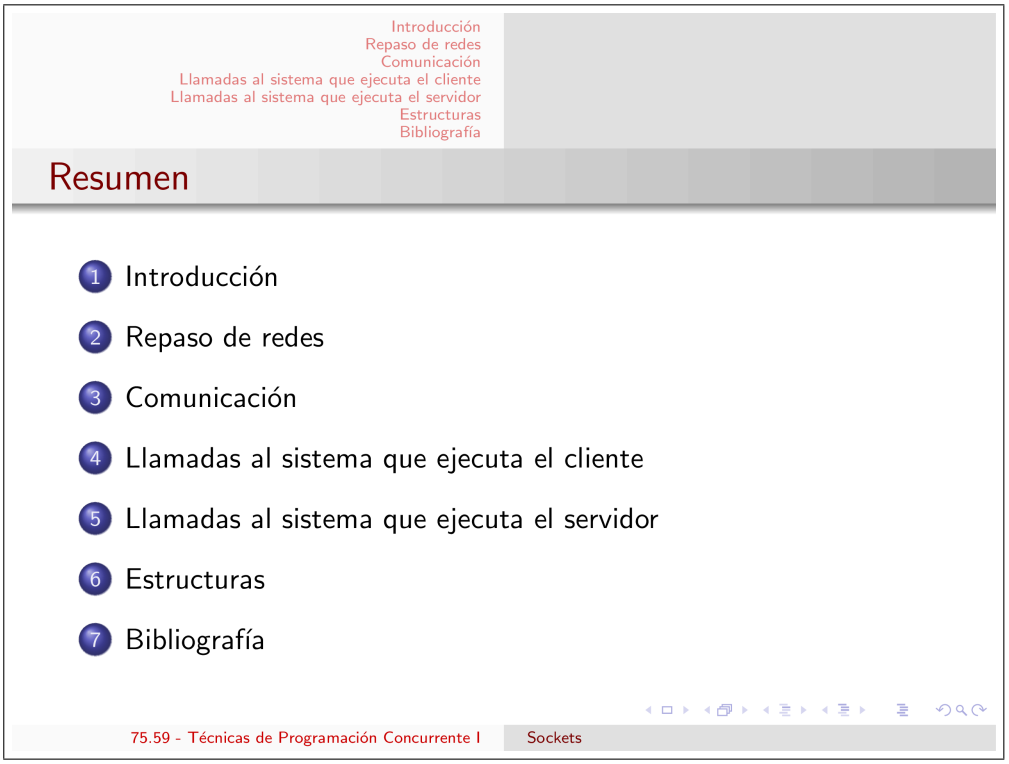
---

## Ejercicios

1. Estudiar la cabecera `<sys/socket.h>`
2. Definir el concepto de *socket* en UNIX.
3. Estudiar las siguientes llamadas al sistema:
   a) *socket*: crea un punto de comunicación
   b) *bind*: relaciona un nombre con un socket
   c) *listen*: dispone un socket para aceptar conxiones y la longitud de la cola
   d) *accept*: acepta una nueva conexión en un socket y crea un nuevo socket
   e) *select*: espera a que la entrada/salida esté lista
4. Escribir un programa tal que el proceso main crea un hijo que actuará como cliente, mientras el padre actúa como servidor. La comunicación se establece para enviar y recibir un saludo, por ejemplo: "Hola hijo" y "Buen día Papá".
5. Modificar el programa del ejercicio anterior para que el servidor pueda gestionar más de un cliente.
6. Buscar otras llamadas relacionadas con la conversión de bits y mostrar cómo se utilizan en un programa.
7. Explicar qué es la "socket address", "AF_UNIX socket address", "AF_INET socket address" y qué estructuras o llamadas al sistema están indicadas para manipular direcciones de *sockets*.
8. ¿Para qué sirve la función *getaddrinfo()*? Indicar cuál es el prototipo y explicar cada uno de los parámetros.
9. Establecer una comparación entre *sockets* y *fifos*.
10. ¿Qué tipos de *sockets* se pueden encontrar en ambiente UNIX?

## Apuntes


Introducción
Repaso de redes
Comunicación
Llamadas al sistema que ejecuta el cliente
Llamadas al sistema que ejecuta el servidor
Estructuras
Bibliografía
Sockets
75.59 - Técnicas de Programación Concurrente I
Facultad de Ingeniería - Universidad de Buenos Aires
75.59 - Técnicas de Programación Concurrente I
Sockets


Introducción
Repaso de redes
Comunicación
Llamadas al sistema que ejecuta el cliente
Llamadas al sistema que ejecuta el servidor
Estructuras
Bibliografía
Resumen
1 Introducción
2 Repaso de redes
3 Comunicación
4 Llamadas al sistema que ejecuta el cliente
5 Llamadas al sistema que ejecuta el servidor
6 Estructuras
7 Bibliografía
75.59 - Técnicas de Programación Concurrente I
Sockets

## Introducción (I)

- Permiten la comunicación entre dos procesos diferentes
  - En la misma máquina
  - En dos máquinas diferentes
- Se usan en aplicaciones que implementan el modelo cliente - servidor:
  - Cliente: es activo porque inicia la interacción con el servidor
  - Servidor: es pasivo porque espera recibir las peticiones de los clientes



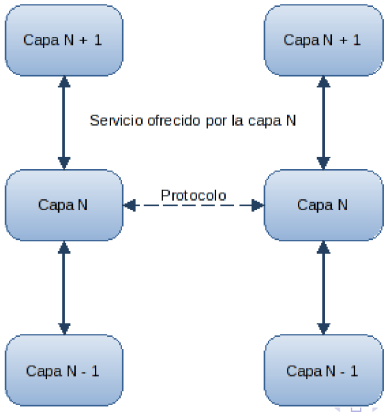


## Introducción (II)

- Arquitectura cliente - servidor
  - Arquitectura de dos niveles: el cliente interactúa directamente con el servidor
  - Arquitectura de tres niveles: middleware
    - Capa de software ubicada entre el cliente y el servidor
    - Provee principalmente seguridad y balanceo de carga
- Tipos de servidor
  - Iterativo: atiende las peticiones de a una a la vez
  - Concurrente: puede atender varias peticiones a la vez

## Repaso de redes (I)

Modelo de capas







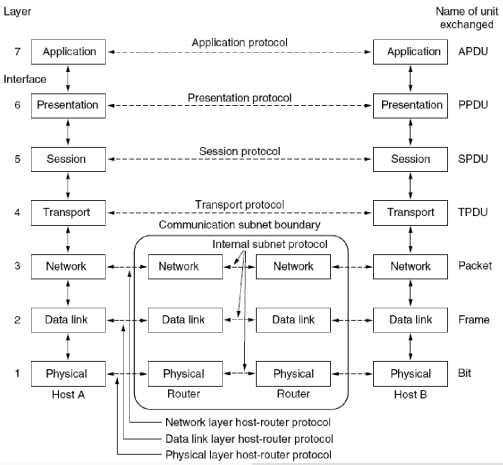

## Repaso de redes (II)

Tipos de servicio:

- Sin conexión
  - Los datos se envían al receptor y no hay control de flujo ni de errores.
- Sin conexión con ACK
  - Por cada dato recibido, el receptor envía un acuse de recibo conocido como *ACK*.
- Con conexión
  - Tres fases: establecimiento de la conexión, intercambio de datos y cierre de la conexión. Hay control de flujo y control de errores.

# Repaso de redes (III)

Modelo OSI





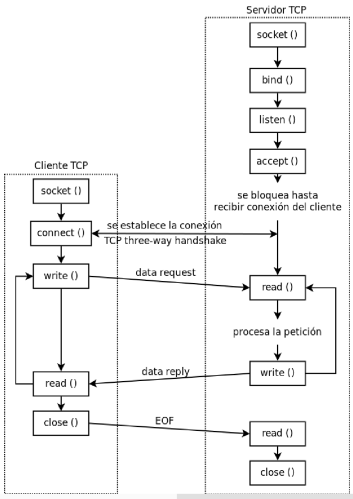



# Tipos de Sockets

Tipos de Sockets

- Stream sockets: usan el protocolo TCP: entrega garantizada del flujo de bytes.
- Datagram sockets: usan el protocolo UDP: la entrega no está garantizada; servicio sin conexión.
- Raw sockets: permiten a las aplicaciones enviar paquetes IP.
- Sequenced packet sockets: similares a *stream sockets*, pero preservan los delimitadores de registro. Utilizan el protocolo SPP (Sequenced Packet Protocol).

## Comunicación








## Creación del socket

Función *socket ()*

```
int socket ( int family,int type,int protocol );
```

- Crea el file descriptor del socket
- Parámetros:
  - Family: AF_INET (IPv4), AF_INET6 (IPv6), AF_UNIX, AF_LOCAL (local)
  - Type: SOCK_STREAM (stream sockets), SOCK_DGRAM (datagram sockets)
  - Protocol: protocolo a utilizar (0, porque normalmente hay un único protocolo por cada tipo de socket)
- Retorna:
  - El file descriptor del socket en caso de éxito (entero positivo)
  - -1 en caso de error, seteando la variable externa *errno*

## Conexión

Función *connect ()*

```
int connect ( int sockfd,struct sockaddr *serv_addr,int addrlen );
```

- Inicia una conexión con el servidor
- Parámetros:
  - sockfd: file descriptor del socket
  - serv_addr: puntero a estructura que contiene dirección IP y puerto destino; se arma con *gethostbyname()* y *bcopy()*
  - addrlen: tamaño de struct sockaddr
- Retorna:
  - 0 en caso de éxito
  - -1 en caso de error, seteando la variable externa *errno*





## Lectura, escritura y cierre del socket

Lectura y escritura

- Función *read()*: lee bytes del socket
- Función *write()*: escribe bytes en el socket
- Funciones *send()* y *recv()*: para comunicación usando stream sockets
- Funciones *sendto()* y *recvfrom()*: para comunicación usando datagram sockets

Cierre

- Función *close()*

## Conexión pasiva (I)

Función *socket ()*: ídem cliente
Función *bind ()*

```
int bind ( int sockfd,struct sockaddr *my_addr,int addrlen );
```

- Asigna una dirección local al socket
- Parámetros:
  - sockfd: file descriptor del socket
  - my_addr: puntero a estructura que contiene dirección IP y puerto local
  - addrlen: tamaño de struct sockaddr
- Retorna:
  - 0 en caso de éxito
  - -1 en caso de error, seteando la variable externa *errno*





## Conexión pasiva (II)

Función *listen ()*

```
int listen ( int sockfd,int backlog );
```

- Convierte un socket sin conectar en socket pasivo
- Parámetros:
  - sockfd: file descriptor del socket
  - backlog: longitud máxima de la cola de conexiones pendientes que puede tener el servidor
- Retorna:
  - 0 en caso de éxito
  - -1 en caso de error, seteando la variable externa *errno*

## Conexión pasiva (III)

Función *accept ()*

```
int accept ( int sockfd,struct sockaddr *cliaddr,socklen_t *addrlen );
```

- Retorna la siguiente conexión completa de la cola de conexiones
- Parámetros:
  - sockfd: file descriptor del socket
  - cliaddr: puntero a estructura con la dirección del cliente
  - addrlen: tamaño de struct sockaddr
- Retorna:
  - El file descriptor del cliente en caso de éxito; se utiliza para comunicarse con el cliente.
  - -1 en caso de error, seteando la variable externa *errno*.





## Lectura, escritura y cierre del socket

Lectura y escritura

- Función *read()*: ídem cliente
- Función *write()*: ídem cliente
- Funciones *send()* y *recv()*: ídem cliente
- Funciones *sendto()* y *recvfrom()*: ídem cliente

Cierre

- Función *close()*

## Estructuras (I)

```
struct sockaddr {
      unsigned short sa_family;
      char sa_data[14];
};
```

Permite guardar información sobre la dirección de un socket.

- sa_family: familia a la cual pertenece la dirección; usaremos únicamente AF_INET
- sa_data: representa la dirección del socket (dirección y puerto); depende del protocolo que se esté utilizando





## Estructuras (II)

```
struct sockaddr_in {
      short int sin_family;
      unsigned short int sin_port;
      struct in_addr sin_addr;
      unsigned char sin_zero[8];
};
```

Solamente para IPv4.

- sin_family: familia de la dirección (AF_INET)
- sin_port: puerto del servicio
- sin_addr: dirección IP (INADDR_ANY)
- sin_zero: se utiliza solamente para que el tamaño coincida con *struct sockaddr* y poder castear

## Estructuras (III)

```
struct in_addr {
      unsigned long s_addr;
};
```

Miembros:

- s_addr: dirección donde escuchará el servidor (INADDR_ANY)





## Bibliografía


- Manuales del sistema operativo
- "Computer Networks", Andrew S. Tanenbaum, cuarta edición
- Tutorial de programación en redes, http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html
- "Unix Network Programming - Volume 1 - The Sockets Networking API", Richard Stevens, tercera edición

# Fuentes de los ejemplos

Listado 1: Cliente

```
#include <iostream>
#include <string.h>
#include "Sockets/ClientSocket.h"


int main ( int argc,char *argv[] ) {

        static const unsigned int BUFFSIZE = 255;
        static const unsigned int SERVER_PORT = 16000;

        if ( argc != 2 ) {
                std::cout << "Uso: ./EchoClient direccion_ip_servidor" << std::endl << std::
                    endl;
                return -1 ;
        }

        try {
                ClientSocket socket ( argv[1],SERVER_PORT );
                char bufferRta[BUFFSIZE];
                char entrada[BUFFSIZE];

                std::cout << "EchoClient: abriendo conexion con el servidor " << argv[1] << std
                    ::endl;
                std::cout << "EchoClient: conexion abierta.  Ingresar el texto a enviar y
                    presionar [ENTER].  s para salir " << std::endl << std::endl;
                socket.abrirConexion();

                std::string mensaje;
                do {
                        std::cin.getline ( entrada,BUFFSIZE );

                        std::cout << "EchoClient: enviando dato al servidor: " << entrada <<
                            std::endl;
                        mensaje = entrada;
                        socket.enviar ( static_cast<const void*>(entrada),mensaje.size() );

                        int longRta = socket.recibir ( static_cast<void*>(bufferRta),BUFFSIZE )
                            ;
                        std::string rta = bufferRta;
                        rta.resize(longRta);

                        std::cout << "EchoClient: respuesta recibida del servidor: " << rta <<
                            std::endl;
                } while ( mensaje != std::string("s") );

                std::cout << "EchoClient: cerrando la conexion" << std::endl;
                socket.cerrarConexion ();
                std::cout << "EchoClient: fin del programa" << std::endl;

        } catch ( std::string& mensaje ) {
                std::cout << mensaje << std::endl;
        }

        return 0;
}
```

Listado 2: Servidor

```
#include <iostream>
#include "Sockets/ServerSocket.h"


int main () {

        static const unsigned int BUFFSIZE = 255;
        static const unsigned int SERVER_PORT = 16000;

        try {
                ServerSocket socket ( SERVER_PORT );
                char buffer[BUFFSIZE];

                std::cout << "EchoServer: esperando conexiones" << std::endl;
                std::cout << "EchoServer: enviar la cadena 's' desde el cliente para terminar"
                    << std::endl << std::endl;
                socket.abrirConexion();

```

```
18                  std::string peticion;
19
20                  do {
21                          int bytesRecibidos = socket.recibir ( static_cast<void*>(buffer),
                                BUFFSIZE );
22                          peticion = buffer;
23                          peticion.resize(bytesRecibidos);
24                          std::cout << "EchoServer: dato recibido: " << peticion << std::endl;
25
26                          std::cout << "EchoServer: enviando respuesta . . ." << std::endl;
27                          socket.enviar ( static_cast<const void*>(peticion.c_str()),peticion.
                                size() );
28                  } while ( peticion != std::string("s") );
29
30                  std::cout << "EchoServer: cerrando conexion" << std::endl;
31                  socket.cerrarConexion ();
32
33          } catch ( std::string& mensaje ) {
34                  std::cout << mensaje << std::endl;
35          }
36
37          return 0;
38  }
```

Listado 3: Clase Socket

```
1   #ifndef SOCKET_H_
2   #define SOCKET_H_
3
4   #include <sys/types.h>
5   #include <sys/socket.h>
6   #include <strings.h>
7   #include <netinet/in.h>
8
9   class Socket {
10
11          protected:
12                  int fdSocket;
13                  struct sockaddr_in serv_addr;
14
15          public:
16                  Socket ( const unsigned int port );
17                  virtual ~Socket ();
18
19                  virtual void abrirConexion () = 0;
20
21                  virtual int enviar ( const void* buffer,const unsigned int buffSize ) const =
                        0;
22                  virtual int recibir ( void* buffer,const unsigned int buffSize ) const = 0;
23
24                  virtual void cerrarConexion () const = 0;
25  };
26
27  #endif /* SOCKET_H_ */
```

Listado 4: Clase Socket

```
1   #include "Socket.h"
2
3   Socket :: Socket ( const unsigned int port ) {
4
5           this->fdSocket = socket ( AF_INET,SOCK_STREAM,0 );
6           if ( this->fdSocket < 0 )
7                   throw "Error al crear el socket";
8
9           // se inicializa la estructura de la direccion
10          bzero ( (char *)&(this->serv_addr),sizeof(this->serv_addr) );
11          this->serv_addr.sin_family = AF_INET;
12          this->serv_addr.sin_port = htons ( port );
13  }
14
15  Socket::~Socket () {
16  }
```

Listado 5: Clase ServerSocket

```
1   #ifndef SERVERSOCKET_H_
2   #define SERVERSOCKET_H_
3
```

```
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <strings.h>
7  #include <arpa/inet.h>
8  #include <netdb.h>
9  #include <string>
10 #include <string.h>
11 #include <errno.h>
12 #include <unistd.h>
13
14 #include "Socket.h"
15
16
17 class ServerSocket : public Socket {
18
19         private:
20                 int nuevoFdSocket;
21                 static const int CONEXIONES_PENDIENTES = 20;
22
23         public:
24                 ServerSocket ( const unsigned int port );
25                 ~ServerSocket ();
26
27                 void abrirConexion ();
28
29                 int enviar ( const void* buffer,const unsigned int buffSize ) const;
30                 int recibir ( void* buffer,const unsigned int buffSize ) const;
31
32                 void cerrarConexion () const;
33 };
34
35 #endif /* SERVERSOCKET_H_ */
```

Listado 6: Clase Socket

```
1  #include "ServerSocket.h"
2
3
4  ServerSocket :: ServerSocket ( const unsigned int port ) : Socket ( port ) {
5          this->nuevoFdSocket = -1;
6  }
7
8  ServerSocket :: ~ServerSocket () {
9  }
10
11 void ServerSocket :: abrirConexion () {
12         struct sockaddr_in cli_addr;
13
14         // el servidor aceptara conexiones de cualquier cliente
15         this->serv_addr.sin_addr.s_addr = INADDR_ANY;
16
17         int bindOk = bind ( this->fdSocket,
18                                                 (struct sockaddr *)&(this->serv_addr),
19                                                 sizeof(this->serv_addr) );
20         if ( bindOk < 0 ) {
21                 std::string mensaje = std::string("Error en bind(): ") + std::string(strerror(
                        errno));
22                 throw mensaje;
23         }
24
25         int listenOk = listen ( this->fdSocket,CONEXIONES_PENDIENTES );
26         if ( listenOk < 0 ) {
27                 std::string mensaje = std::string("Error en listen(): ") + std::string(strerror
                        (errno));
28                 throw mensaje;
29         }
30
31         int longCliente = sizeof ( cli_addr );
32
33         this->nuevoFdSocket = accept ( this->fdSocket,
34                                                                         (struct sockaddr *)&
                                                                            cli_addr,
35                                                                         (socklen_t *)&
                                                                            longCliente );
36
37         if ( this->nuevoFdSocket < 0 ) {
38                 std::string mensaje = std::string("Error en accept(): ") + std::string(strerror
                        (errno));
39                 throw mensaje;
40         }
41 }
```

```
42
43 int ServerSocket :: enviar ( const void* buffer,const unsigned int buffSize ) const {
44         int cantBytes = write ( this->nuevoFdSocket,buffer,buffSize );
45         return cantBytes;
46 }
47
48 int ServerSocket :: recibir ( void* buffer,const unsigned int buffSize ) const {
49         int cantBytes = read ( this->nuevoFdSocket,buffer,buffSize );
50         return cantBytes;
51 }
52
53 void ServerSocket :: cerrarConexion () const {
54         close ( this->nuevoFdSocket );
55         close ( this->fdSocket );
56 }
```

Listado 7: Clase ClientSocket

```
1  #ifndef CLIENTSOCKET_H_
2  #define CLIENTSOCKET_H_
3
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <strings.h>
7  #include <arpa/inet.h>
8  #include <netdb.h>
9  #include <string>
10 #include <string.h>
11 #include <errno.h>
12 #include <unistd.h>
13
14 #include "Socket.h"
15
16 class ClientSocket : public Socket {
17
18         private:
19                 std::string ipServidor;
20
21         public:
22                 ClientSocket ( const std::string& ipServidor,const unsigned int port );
23                 ~ClientSocket ();
24
25                 void abrirConexion ();
26
27                 int enviar ( const void* buffer,const unsigned int buffSize ) const;
28                 int recibir ( void* buffer,const unsigned int buffSize ) const;
29
30                 void cerrarConexion () const;
31 };
32
33 #endif /* CLIENTSOCKET_H_ */
```

Listado 8: Clase Socket

```
1  #include "ClientSocket.h"
2
3  ClientSocket :: ClientSocket ( const std::string& ipServidor,const unsigned int port ) : Socket
       ( port ) {
4          this->ipServidor = ipServidor;
5  }
6
7  ClientSocket :: ~ClientSocket () {
8  }
9
10 void ClientSocket :: abrirConexion () {
11
12         struct hostent *server = gethostbyname ( this->ipServidor.c_str() );
13         if ( server == NULL ) {
14                 std::string mensaje = std::string("No se puede localizar el host: ") + std::
                       string(strerror(errno));
15                 throw mensaje;
16         }
17
18     bcopy ( (char *)server->h_addr,
19                 (char *)&(this->serv_addr.sin_addr.s_addr),
20                 server->h_length );
21
22     int connectOk = connect ( this->fdSocket,
23                                                 (const struct sockaddr *)&(this->serv_addr),
24                                                 sizeof(this->serv_addr) );
```

```
25      if ( connectOk < 0 ) {
26          std::string mensaje = std::string("Error en connect(): ") + std::string(strerror(errno)
                );
27          throw mensaje;
28      }
29  }
30
31  int ClientSocket :: enviar ( const void* buffer,const unsigned int buffSize ) const {
32          int cantBytes = write ( this->fdSocket,buffer,buffSize );
33          return cantBytes;
34  }
35
36  int ClientSocket :: recibir ( void* buffer,const unsigned int buffSize ) const {
37          int cantBytes = read ( this->fdSocket,buffer,buffSize );
38          return cantBytes;
39  }
40
41  void ClientSocket :: cerrarConexion () const {
42          close ( this->fdSocket );
43  }
```