Interfaz de Socket

Agustín J. González

ELO309

Introducción

- ¿Cómo las aplicaciones se comunican con la familia de protocolos de software y así logran comunicación con aplicaciones remotas?
- La interfaz de programas de aplicación usada para inter-actuar con los protocolos de red se conocen como las APIs para la programación de red. La más conocida y difundida es la **API de Socket** (Socket Application Program Interface).
- A través de la interfaz de socket, las aplicaciones especifican los detalles de la comunicación como: qué protocolo se usará, es una aplicación cliente o servidor, y máquina remota.
- En el sistema operativo UNIX esta API es parte del SO. Otros sistemas operativos han creado una biblioteca (*library*) que implementa todos los llamados de la API de socket y así el código UNIX puede ser fácilmente compilado y ejecutado.

Modelo de comunicación de Socket y I/O en UNIX

- El modelo empleado en socket para la comunicación entre aplicaciones remotas es similar al usado en la comunicación con cualquier dispositivo de I/O en UNIX. Enviar datos a una aplicación remota usa el mismo tipo de llamados que enviar datos a un archivo.
- En UNIX cuando se abre un archivo se retorna un descriptor de archivo. Cuando se abre un socket también se retorna un descriptor.
- Un descriptor es un entero. El SO usa este entero para entrar en una tabla que contiene todos los atributos del dispositivo al cual se refiere el descriptor.

Parámetros usados en la API de Sockets

- Las mayores diferencias entre la comunicación vía sockets y manejo de archivos es la la forma de abrir el canal.
- Para ***crear*** un sockets la aplicación debe especificar :
  - Protocolo de transporte (su familia y protocolo dentro de la familia)
  - Dirección de la máquina remota o dirección local donde se esperará por requerimientos remotos,
  - Es una cliente o un servidor,
  - El puerto asociado a la aplicación

Creación y cierre de un socket

- Creación:
  int descriptor;
  descriptor = socket(protocolFamily, type, protocol);
  donde:
  - Familia de protocolo: PF_INET, PF_APPLETALK
  - Tipo: SOCK_STREAM para servicios de conexión y SOCK_DGRAM para servicios sin conexión.
  - Protocolo: El protocolo particular a ser usado de la familia de protocolo. En TCP/IP, éste puede ser tcp o udp.
- Cierre del un socket:
  close(descriptor);
  Se termina la conexión antes de cerrar el socket.

Procedimientos en servidor

- Procedimiento **bind**
  Este procedimiento asocia o liga un socket con una dirección IP y un puerto local.
  bind(descriptor, local_addr, addr_len);
  - Descriptor: es el descriptor del socket.
  - Dirección local: es una estructuraconteniendo IP local y puerto local.
  - Largo de dirección: entero que especifica el largo de la estructura local_addr.
- Como el llamado debe servir para varios protocolos, la estructura en genérica:

Procedimientos en servidor (cont..)

- struct sockaddr {
  u_char sa_len; // largo total
  u_char sa_family; // familia de la dirección
  char sa_data[14]; //la dirección
  }
- En el caso particular de TCP/IP el campo dirección está especificado como sigue:
- struct sockaddr_in {
  u_char sin_len; // largo total
  u_char sin_family; // familia de la dirección
  u_short sin_port; // número de puerto
  struct in_addr sin_addr; // dirección IP
  char sin_zero[8]; // no usados
  }

Procedimientos en servidor (cont..)

- Procedimiento **listen**
  listen(descriptor, queue_size);
  Instruye al OS que el socket es pasivo y desea tener a lo más queue_size requerimientos de conexión pendientes.
  - El sistema rechaza requerimientos cuando la cola se llena.
- Procedimiento **accept**
  newDescriptor = **accept**(descriptor, c_address, c_address_len);
  - Esta llamada es usada por servidores que usan servicios de conexión.
  - c_address es del tipo struct sockaddr ya visto.
  - Luego de aceptar una conexión, el servidor atiende al cliente a través del descriptor especialmente creado para él. Mientras tanto el servidor puede aceptar nuevas conexiones en el descriptor original.

Procedimientos en cliente

- Procedimiento **connect**
  connect(descriptor, srv_address, srv_address_len);
  - srv_address: es una estructura del tipo struct sockaddr que contiene la dirección del servidor.
  - srv_address_len es el largo de la estructura.
- Cuando el servicio es de conexión, connect establece la conexión con el servidor, el cual debe aceptar con accept().

Procedimientos de envío y recibo

- Procedimiento **send**
  int send (descriptor, data, length, flags);

  data es la dirección de memoria donde se encuentran los datos a enviar.
  Length: es el largo en bytes de la zona de datos. Ésta función retorna el número de bytes efectivamente enviados
  Flags: especifican opciones. Las aplicaciones normales no las usan. Son usadas por programas de monitoreo y debug.
- Procedimiento **sendto**
  int sendto (descriptor, data, length, flags, dest_address, address_len);

Procedimientos de envío y recibo

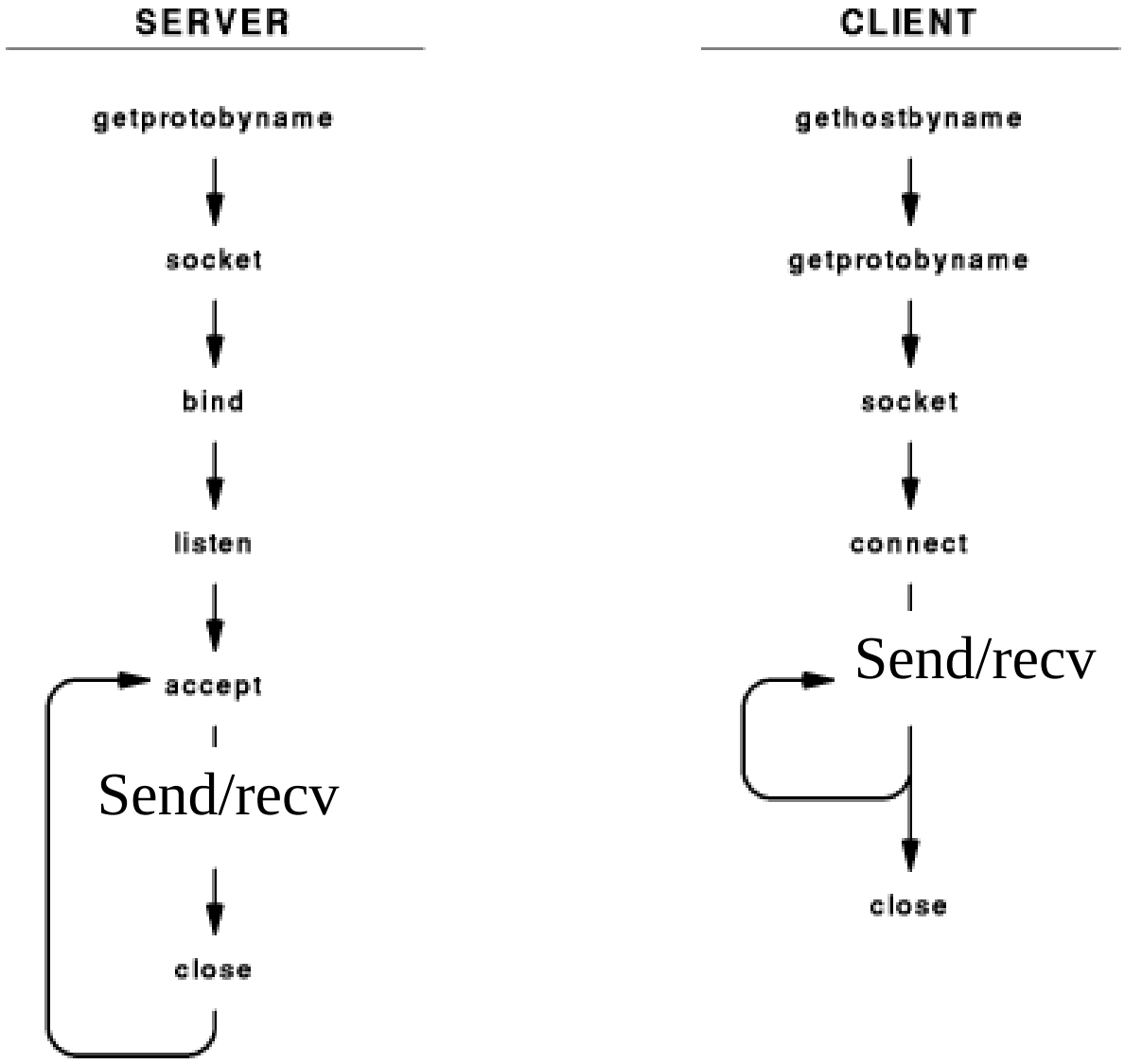
- Procedimiento **recv**
  int recv (descriptor, buffer, length, flags);

  buffer es la dirección de memoria donde se deben depositar los datos a recibir.
  length: es el largo en bytes de los datos que se desean leer. **Recv** retorna el número de bytes efectivamente leídos.
  Flags: especifican opciones. Las aplicaciones normales no las usan. Son usadas por programas de monitoreo y debug.
- Procedimiento **recvfrom**
  int recvfrom (descriptor, buffer, length, flags, source_address, address_len);

Otras funciones relacionadas

- Procedimientos **getpeername**: permite obtener la dirección y puerta remota a partir de un socket ya “conectado”.
- Procedimiento **gethostname**: permite obtener el nombre de la máquina local.
- Procedimientos **getsockopt** y **setsockopt**: permiten obtener y modificar atributos de un socket; por ejemplo, el tramaño del buffer usado por TCP para recibir paquetes.
- Procedimientos **gethostbyname** y **gethostbyaddr**: permiten obtener la información sobre una máquina a partir de su nombre o dirección IP.

Cliente/Servidor TCP: Posible Secuencia de Eventos



Cliente/Servidor UDP: Principios de voz sobre IP

- Mic: aplicación que lee muestras de audio desde el dispositivo de audio y las envía a la salida estándar del programa (pantalla normalmente)
- speaker: aplicación que toma muestras de la entrada estándar (normalmente teclado) y los envía al parlante.
- Una “pipe” en Unix conecta la salida estándar de un programa con la entrada de otro.
- Mic | speaker hace que lo enviado a la salida estándar no vaya a pantalla sino que ingrese como entrada estándar para speaker
- Si combinamos estas aplicaciones con las cliente/servidor vistas podemos enviar audio en internet (muy básico ....)
- En el transmisor: mic | UDPclient localhost 2345
- En el receptor: UDPserver 2345 | speaker
- Esto también se puede hacer entre máquinas en Internet.