SISTEMAS OPERATIVOS II

TEMA 4. IPC: Comunicación entre Procesos

Área de Arquitectura y Tecnología de Computadores
Escuela Universitaria Politécnica de Teruel
http://????.????.es/SOII/

IPC: Comunicación entre Procesos

- **Objetivos**:
  - Entender el concepto de Comunicación Entre Procesos: IPC
  - Conocer y entender los principales mecanismos de IPC en Unix basados en un SPM:
    - Sockets
    - Llamadas a procedimientos remotos: RPC

IPC: Comunicación entre Procesos

- Introducción
- Mecanismos de IPC en Unix
  - Programación Cliente / Servidor en C mediante sockets
  - Programación Cliente / Servidor en C mediante RPC

Introducción

- **IPC: “Interprocess Communication”**
  - Dos tipos de comunicación:
    - Basada en variables compartidas
      - Requiere que el usuario introduzca código para gestionar los buffers de comunicación
      - **Inconveniente**: Los procesos deben residir en la misma máquina
    - Basada en el paso de mensajes
      - Permite comunicar procesos sin recurrir a variables compartidas
      - **Ventaja**: Los procesos pueden residir en diferentes máquinas

Introducción

- **IPC: “Interprocess Communication”**
  - El IPC proporciona un mecanismo que permite la comunicación y sincronización entre procesos mediante el paso de mensajes
  - Estructura básica del IPC:
    - Operación send(mensaje)
    - Operación receive(mensaje)
  - **Nota**: Los mensajes pueden ser de longitud fija o variable

Introducción

- **IPC: “Interprocess Communication”**
  - Proceso de comunicación:
    - Establecer el enlace de comunicación o link
      - Realización física: memoria compartida, bus, red, etc.
      - Realización lógica
    - Intercambiar mensajes vía operaciones send / receive

Introducción

- **Ejemplo de IPC:**
  - El problema del Productor/Consumidor

Proceso productor

```
repeat
    . . .
produce item en var nextp
    . . .
send(consumidor, nextp);
    . . .
until false;
```

Proceso consumidor

```
repeat
    . . .
receive(productor, nextc);
    . . .
consume item en var nextc
    . . .
until false;
```


Introducción

- **Cuestiones para la realización de un mecanismo de IPC:**
  - ¿Cómo se establecen los enlaces de comunicación?
  - ¿Puede un enlace de comunicación estar asociado a más de dos procesos?
  - ¿Cuántos enlaces de comunicación puede haber entre dos procesos?
  - ¿Qué capacidad tiene el enlace de comunicación?
  - ¿Cuál es el tamaño del mensaje?, ¿mensajes de tamaño fijo o mensajes de tamaño variable?
  - ¿El enlace de comunicación es “uni-” o “bi-” direccional?

Introducción

- **Características de los enlaces de comunicación:**
  - Comunicación directa o indirecta
  - Comunicación simétrica o asimétrica
  - Especificación de los buffers de comunicación explícita o automática
- **Características de las operaciones send / receive:**
  - Operación receive:
    - Bloqueadora / NO Bloqueadora
  - Operación send
    - Bloqueadora / NO Bloqueadora
    - Por copia o por referencia

Introducción

- **IPC: Comunicación directa**
  - Los procesos deben especificar el nombre del proceso con el que se quieren comunicar
    - send(P, mensaje); enviar mensaje a P
    - receive(Q, mensaje); recibe mensaje de Q
  - Propiedades del enlace:
    - Los enlaces de comunicación se establecen automáticamente, conociendo las identidades
    - Un enlace de comunicación está asociado exactamente con dos procesos
    - Entre dos procesos sólo existe un enlace de comunicación
    - El enlace de comunicación puede ser uni-direccional, aunque normalmente es bi-direccional

Introducción

- **Ejemplo de IPC: Comunicación directa**
  - El problema del Productor/Consumidor

    (Simetría de direccionamiento)

Proceso productor

```
repeat
   . . .
produce item en var nextp
   . . .
send(consumidor, nextp);
   . . .
until false;
```

Proceso consumidor

```
repeat
   . . .
receive(productor, nextc);
   . . .
consume item en var nextc
   . . .
until false;
```


Introducción

- **Ejemplo de IPC: Comunicación directa**
  - El problema del Productor/Consumidor

    (Asimetría de direccionamiento)

Proceso productor

```
repeat
   . . .
produce item en var nextp
   . . .
send(consumidor, nextp);
   . . .
until false;
```

Proceso consumidor

```
repeat
   . . .
receive(identificador, nextc);
   . . .
consume item en var nextc
   . . .
until false;
```

Introducción

- **IPC: Comunicación indirecta**
  - Los mensajes son enviados y recibidos a través de mailboxes o puertos
    - **Def.-** Un ***mailbox*** es un objeto abstracto donde los procesos depositan y extraen mensajes
    - Cada mailbox tiene un identificador único
    - Dos procesos pueden comunicarse sólo si comparten un mailbox
  - Primitivas para la comunicación:
    - send(A, mensaje); enviar mensaje al mailbox A
    - receive(A, mensaje); recibe mensaje del mailbox A
    - Crear / Destruir un mailbox

Introducción

- **IPC: Comunicación indirecta**
  - Propiedades del enlace de comunicación:
    - Los enlaces de comunicación se establecen solamente si los dos procesos comparten un mailbox
    - Un enlace de comunicación puede estar asociado con muchos procesos
    - Cada pareja de procesos puede compartir varios enlaces de comunicación
    - El enlace de comunicación puede ser uni-direccional o bi-direccional

Introducción

- **IPC: Comunicación indirecta**
  - **"El problema de compartir los enlaces de comunicación"**

    P1, P2 y P3 comparten el mailbox A

    P1 realiza una operación send a A

    P2 y P3 realizan una operación receive de A
    - ¿Quién obtiene el mensaje?
    - **Soluciones:**
      - Permitir que un enlace de comunicación esté asociado como máximo con dos procesos
      - Permitir que como máximo un proceso ejecute una operación receive
      - Permitir al sistema seleccionar arbitrariamente qué proceso recibirá el mensaje: P2 o P3 pero no ambos

Introducción

- **IPC: El enlace de comunicación**
  - **Def.-** ***Capacidad de un enlace de comunicación*** es el número de mensajes que puede contener temporalmente
  - El **Buffering** se puede ver como una cola de mensajes asociada al enlace de comunicación:
    - Con capacidad 0
      - Los procesos deben sincronizarse para que la transferencia tenga lugar: randezvous – comunicación síncrona
    - Con capacidad limitada: longitud máxima = n
      - Relación Productor / Consumidor: el proceso que envía, si el enlace está lleno, debe esperar
    - Con capacidad ilimitada: longitud máxima = ∞
      - El proceso que envía nunca debe esperar

Introducción

- **IPC: El enlace de comunicación. Problemas**
  - En enlaces de comunicación con capacidad no-nula:
    - El proceso que envía no sabe cuando ha sido leído su mensaje: comunicación asíncrona
  - Cuando ocurre un fallo debe entrar en acción un recuperador de errores: “exception condition handling”
  - Condiciones de excepción relacionadas con mensajes:
    - Terminación de procesos
    - Algunos mensajes nunca serán recibidos
    - Algunos procesos quedarán a la espera de mensajes que nunca llegarán

Introducción

- **IPC: El enlace de comunicación. Problemas**
  - Pérdida/Corrupción de mensajes
    - Detección de la pérdida de mensajes:
      - "time out"
      - códigos de error
    - Técnicas de retransmisión
    - Técnicas de notificación

IPC en Unix: sockets

- La familia de protocolos TCP/IP
- Nivel de red: IP
- Nivel de transporte: UDP y TCP
- La abstracción socket: interfaz con TCP/IP
- Sockets: Llamadas al sistema
- Sockets: Funciones interesantes
  - Obtener información de la máquina
  - Obtener información de la red
  - Obtener información de los protocolos
  - Otras funciones

IPC en Unix: sockets

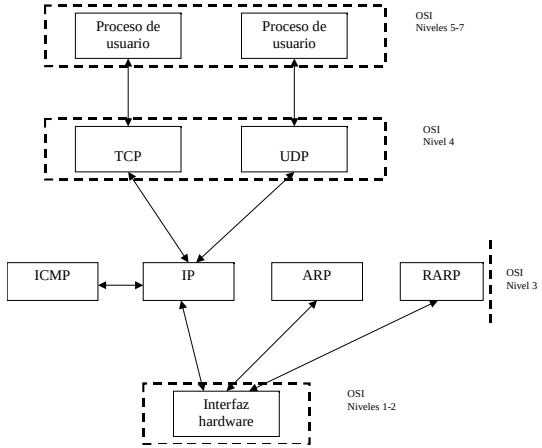
- **La familia de protocolos TCP/IP**
  - Internet es una red que ha sido diseñada sin respetar el modelo OSI
  - La familia de protocolos TCP/IP fue creada a finales de los 60. El modelo OSI todavía no se había definido
  - Niveles OSI:
    1.- Nivel Físico
    2.- Nivel de Enlace
    3.- Nivel de Red
    4.- Nivel de Transporte
    5.- Nivel de Sesión
    6.- Nivel de Presentación
    7.- Nivel de Aplicación

IPC en Unix: sockets

- **La familia de protocolos TCP/IP:** Niveles de la familia TCP/IP (Internet)
  - Data Link Layer: niveles 1, 2 de OSI
  - Internet Protocol (IP): nivel 3 de OSI
  - Transmisión Control Protocol (TCP) y User Datagram Protocol (UDP): nivel 4 de OSI
  - Servicios de usuario ofrecidos por la familia TCP/IP : niveles 5 - 7 de OSI
    - Transferencia de ficheros TCP (FTP)
    - Correo electrónico (SMTP)
    - Conexión remota (TELNET)

IPC en Unix: sockets

- **La familia de protocolos TCP/IP:**



IPC en Unix: sockets

- **La familia de protocolos TCP/IP:** Lista de acrónimos
  - **ICMP**: Internet Control Message Protocol
  - **ARP**: Address Resolution Protocol
    - Mapeo de la @ Internet en la @ hardware
  - **RARP**: Reverse Address Resolution Protocol
    - Mapeo de la @ hardware en la @ Internet
  - Notar que ARP y RARP se usan para buscar la @ Internet dada la @ hardware y viceversa

IPC en Unix: sockets

- **Nivel de red: IP**
  - El protocolo IP de nivel de red nos proporciona un servicio sin conexión NO seguro
  - Servicio sin conexión ≅ servicio postal
    @origen @destino
  - Los mensajes pueden recibirse en orden distinto al orden de envío
  - Servicio NO seguro:
    - No garantiza la llegada de todos los mensajes a su destino
  - IP acepta mensajes de cualquier tamaño < 64 Kbytes
    - Si es necesario, IP troceará estos mensajes y luego los unirá

IPC en Unix: sockets

- **Nivel de red: IP**
  - Cada familia de protocolos define su direccionamiento para identificar las máquina dentro de la red
    - @ Internet utiliza 32 bits y codifica un identificador de red y un identificador de host relativo a esa red: netid / hostid
  - Cada host de Internet debe tener una dirección de 32 bits única
    - @ Internet: 1. 2 . 255 . 4
  - NIC: Network Information Center

IPC en Unix: sockets

- **Nivel de red: IP:** ¿Cómo programar una transferencia entre dos sistemas que sólo ofrecen IP?
  - Dos funciones, **enviar** y **recibir**, a las que pasaríamos las direcciones origen y destino del mensaje
  - Si el tamaño es grande, deberíamos trocear nuestro mensaje
  - La aplicación debería estar preparada para reordenar los trozos a la llegada
    - IP no asegura que el orden de emisión sea el mismo que el de recepción
  - La aplicación debería tener algún método de detección y corrección de errores
  - Necesitaremos implementar algún método para identificar varios destinos dentro de una máquina

IPC en Unix: sockets

- **Nivel de transporte: UDP y TCP:** Servicio UDP, User Datagram Protocol
  - Servicio no orientado a conexión ≅ servicio postal
  - Ofrece los mismos servicio que IP
  - Ofrece la posibilidad de detección de errores
    - **! !** aunque podemos perder mensajes enteros, UDP no nos avisa
  - Permite gestionar los números de puertos permitiendo varias comunicaciones simultáneas en la misma máquina

IPC en Unix: sockets

- **Nivel de transporte: UDP y TCP:** Servicio TCP, Transmission Control Protocol
  - Servicio orientado a conexión ≅ servicio telefónico
  - Ofrece primitivas para establecer la conexión y desconexión
  - Ofrece primitivas de envío y recepción
  - Acepta mensajes de longitud arbitraria
  - Garantiza el orden de los paquetes: mensaje enviados a trozos
  - Entrega los mensajes completos y libre de errores
  - Utiliza números de 16 bits para identificar el destino en una misma máquina: **port numbers**

IPC en Unix: sockets

- **Nivel de transporte: UDP y TCP:** Requisitos para definir una comunicación sobre el nivel de transporte
  - Definición del protocolo de transporte a utilizar: UDP o TCP
  - Definición de la dirección de la máquina local en Internet
  - Definición del número de puerto en la máquina local
  - Definición de la dirección de la máquina remota en Internet
  - Definición del número de puerto en la máquina remota

  {protocolo, @host-local, port-local, @host-remoto, port-remoto}
  - Notar que los detalles del interfaz TCP/IP dependen de la arquitectura del S. O. que los implementa

IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**
  - El interfaz de sockets fue creado en el entorno BSD de Unix
  - Fue creado como un interfaz usuario / programador para usar los servicios de red de las familias de protocolos:
    - **UNIX**
    - **INTERNET:** TCP/UDP
    - **XEROX:** XNS
  - El interfaz tiene todas las primitivas necesarias para la correcta y total utilización de los servicios que ofrecen estos protocolos
  - Las aplicaciones basadas en sockets son fácilmente portables de un sistema a otro

IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**

  El modelo cliente / servidor
  - A través de TCP/IP y del interfaz socket se pueden desarrollar aplicaciones para intercambiar información entre procesos:
    - Que residan en la misma máquina
    - Que residan en máquinas remotas
  - Toda aplicación distribuida se divide en dos partes:
    - **Un cliente**
      - Programa que inicia la comunicación con el servidor
    - **Un servidor**
      - Programa que espera la solicitud de un servicio por parte del cliente

IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**

  El modelo cliente / servidor
  - **Problema**: La sincronización
    - El problema de la sincronización aparece cuando se quieren comunicar dos aplicaciones que se ejecutan en máquinas diferentes
  - **Solución**:
    - Asegurar que una de ellas arranca su ejecución y espera indefinidamente a que la otra contacte con ella
  - **Nota**:
    - Los servidores necesitan acceder a información, puertos, recursos, etc. que son de acceso restringido dentro del S. O., por lo tanto, necesitarán permisos especiales en el sistema

IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**
  El modelo cliente / servidor
  - Tipos de modelos de clientes:
    - **Estándar:**
      - TELNET (23), FTP (21), E-MAIL(25) …
    - **No estándar:**
      - Resto de aplicaciones que sólo están disponibles en esa máquina

IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**
  El modelo cliente / servidor
  - Tipos de modelos de servidores:
    - **Con conexión: TCP**
    - **Sin conexión: UDP**
    - **Iterativo**
    - **Concurrente**

IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**
  El modelo cliente / servidor
  - Modelo de servidor Con conexión: **TCP**
    - TCP proporciona toda la seguridad necesaria para comunicar a través de una red:
      - Verifica la recepción de los datos
      - Retransmisiones
      - Checksum sobre los datos
      - Número de secuencia. Orden de la información
      - Control de flujo
      - Informa del estado de la red

IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**
  El modelo cliente / servidor
  - Modelo de servidor Sin conexión: **UDP**
    - UDP no garantiza una entrega fiable de los mensajes
    - Un mensaje puede perderse, llegar duplicado o modificado por la red sin que UDP lo pueda saber

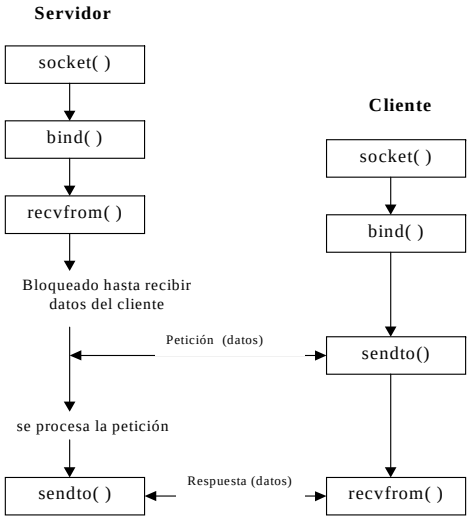
IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**
  El modelo cliente / servidor
  - Modelo de servidor Iterativo:
    - Sirve peticiones en serie: una tras otra
    - Sólo se utilizan si la respuesta puede ser elaborada en un tiempo conocido no muy largo
  - Modelo de servidor Concurrente:
    - Cada vez que llega una nueva petición se crea un proceso hijo que la atiende
    - Los nuevos hijos servirán la petición directamente y finalizarán
    - El proceso padre siempre permanecerá esperando nuevas peticiones

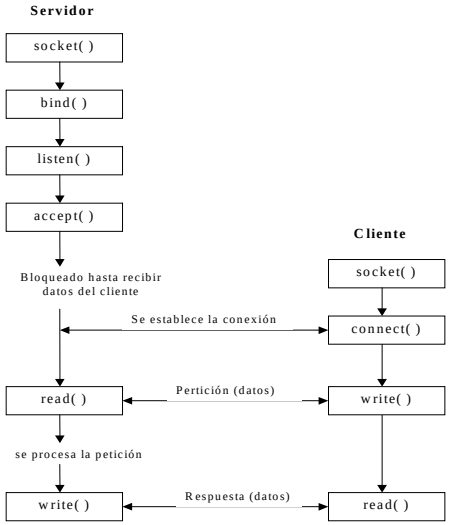
IPC en Unix: sockets

- **La abstracción socket: Interfaz con TCP/IP**
  El modelo cliente / servidor
  - **Nota**:
    - Si queremos cierta seguridad en nuestras aplicaciones y utilizamos UDP se deben tomar las precauciones oportunas para la detección y recuperación de los posibles errores que se puedan producir

Sockets con UDP: Llamadas al sistema



Sockets con TCP: Llamadas al sistema



IPC en Unix: sockets

- **Llamada al sistema *socket***: creación de un socket
  - socket - create an endpoint for communication
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - int socket (int domain, int type, int protocol);
  - Devuelve el descriptor asociado al socket, servirá para hacer referencia al socket en las llamadas
  - Argumentos de la llamada:
    - **domain**
    - **type**
    - **protocol**

IPC en Unix: sockets

- **Llamada al sistema *socket***: creación de un socket
  - Argumentos de la llamada:
    - **domain:** familia de protocolos que se desea usar en la comunicación:
      - AF_UNIX protocolos internos UNIX
      - AF_INET protocolos de Internet
      - AF_NS protocolos de Xerox NS
      - AF_APPLETALK protocolos con red Appletalk
    - **type:** indica el tipo de comunicación que se desea
      - En AF_INET tenemos:
        - SOCK_STREAM TCP
        - SOCK_DGRAM UDP
        - SOCK_RAW ICMP

IPC en Unix: sockets

- **Llamada al sistema *socket***: creación de un socket
  - Argumentos de la llamada:
    - **protocol:** identifica un protocolo concreto para el tipo de socket

Familia	Tipo	Protocolo	Prot. asociado
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UPD
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPPROTO_RAW	(raw)

IPC en Unix: sockets

- **Llamada al sistema *bind***: asignación de una @ local
  - bind - bind a name to a socket
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - int bind (int sockfd, const struct sockaddr *myaddr, int addrlen);
  - Asigna una determina dirección a un socket
  - Argumentos de la llamada:
    - **sockfd**
    - **myaddr**
    - **addrlen**

IPC en Unix: sockets

- **Llamada al sistema *bind***: asignación de una @ local
  - Argumentos de la llamada:
    - **sockfd:** descriptor asociado al socket
    - **myaddr:** puntero a una estructura que contiene la dirección del socket: protocolo, dirección IP y puerto
    - **addrlen:** longitud, en bytes, de la estructura apuntada por myaddr
    - **struct sockaddr**

```
{
    u_short sa_family;    /* familia de direcciones a usar */
    char sa_data[14];     /*  14 bytes para la dirección del
                              protocolo dado */
};
```


IPC en Unix: sockets

- **Llamada al sistema *bind***: asignación de una @ local
  - ¿Cómo se utiliza?
    - Un servidor tiene que registrar en el sistema la dirección que conocen sus clientes
    - Cualquier servidor debe hacer esta llamada antes de aceptar peticiones de un cliente
    - Un cliente sin conexión quiere asegurase de que el sistema le asigne una dirección única para que el servidor sepa donde devolver la información

IPC en Unix: sockets

- **Llamada al sistema *bind***: asignación de una @ local
  - Otras estructuras para TCP/IP (#include <netinet/in.h>)
    - **struct in_addr**

```
{
    u_long s_addr;        /* 32 bits para la dirección IP */
};
```

    - **struct sockaddr_in**

```
{
    short sin_family;       /* AF_INET */
    u_short sin_port;       /* 16 bits para el número de puerto */
    struct in_addr sin_addr;    /* 32 bits para la dirección IP */
    char sin_zero[8];       /* no usados */
};
```


IPC en Unix: sockets

- **Llamada al sistema *bind***: asignación de una @ local
  - Otras estructuras para TCP/IP (#include <netinet/in.h>)
    - Notar que una vez definida la dirección en una variable de tipo **sockaddr_in,** podemos utilizarla como argumento en las funciones haciendo una conversión al tipo **sockaddr**

IPC en Unix: sockets

- **Llamada al sistema *connet***: establecimiento de la conexión
  - connect - initiate a connection on a socket
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - int connect(int sockfd, struct sockaddr *servaddr, int addrlen);
  - Se utiliza para establecer una conexión con otro proceso
  - Argumentos de la llamada
    - **sockfd**
    - **servaddr**
    - **addrlen**

IPC en Unix: sockets

- **Llamada al sistema *connet***: establecimiento de la conexión
  - Argumentos de la llamada
    - **sockfd:** descriptor asociado al socket
    - **servaddr:** dirección destino a la que queremos conectar el socket
    - **addrlen:** entero que indica la longitud, en bytes, de la dirección destino

IPC en Unix: sockets

- **Llamada al sistema *connet***: establecimiento de la conexión
  - Clientes UDP**:**
    - Pueden llamar a **connect( )**
      - Lo único que hará será registrar la dirección del socket del proceso remoto
    - El cliente sólo recibirá datagramas de la dirección especificada
    - Un cliente que llame a **connect( )** no necesitará especificar la dirección del socket remoto a la hora de enviar o recibir mensajes:
      - Por tanto puede utilizar: read, write, recv, send
    - Podemos validar la dirección del socket ya que en caso de error el usuario será avisado inmediatamente

IPC en Unix: sockets

- **Llamada al sistema *connet***: establecimiento de la conexión
  - Clientes TCP:
    - **connect** le sirve al cliente para realizar una petición de conexión al servidor
    - La petición será atendida si el servidor esta bloqueado en **accept( )**
    - La petición será encolada si la cola estaba vacía o el servidor no acepta la conexión en ese momento
  - Notar que un cliente UDP no necesita utilizar **bind( )** antes de **connect( )** ya que esta última se encarga de asociar tanto la dirección del socket local como la dirección del socket remoto

IPC en Unix: sockets

- **Llamada al sistema *listen***: Se indica al sistema que ya se pueden recibir peticiones de conexión
  - listen - listen for connections on a socket
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - int listen(int sockfd, int glength);
  - Se utiliza para que un proceso servidor orientado a conexión indique al sistema que está preparado para recibir peticiones de conexión
  - Argumentos de la llamada
    - **sockfd**: descriptor asociado al socket
    - **glength:** longitud de la cola de peticiones del socket, el valor máximo soportado por el S. O. es 5

IPC en Unix: sockets

- **Llamada al sistema *accept***: Aceptar posibles peticiones de los clientes
  - accept - accept a connection on a socket
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - int accept(int sockfd, struct sockaddr *addr, int *addrlen);
  - Devuelve el descriptor de un nuevo socket que será utilizado para la comunicación
  - Argumentos de la llamada
    - **sockfd**
    - **addr**
    - **addrlen**

IPC en Unix: sockets

- **Llamada al sistema *accept***: Aceptar posibles peticiones de los clientes
  - La utiliza el servidor para atender las posibles peticiones de los clientes
  - Si no hay ninguna petición pendiente en la cola, el proceso quedará bloqueado hasta que llegue una nueva petición
  - La estructura **addr** se actualiza con la dirección del socket cliente que pidió la conexión y **addrlen** con el tamaño en bytes de esa dirección
    - Estos datos podrían ser usados para conocer la identidad del cliente, aceptar o rechazar el servicio, etc.
  - Cuando dos procesos están conectados, ya pueden enviar y recibir datos

IPC en Unix: sockets

- **Llamada al sistema *read***: Transferencia de datos
  - read - read from socket
    - #include <unistd.h>
    - int read(int sockfd, char *buffer, int length);
  - Devuelve el número de bytes recibidos por el socket
  - Argumentos de la llamada
    - **sockfd:** descriptor asociado al socket
    - **buffer:** @ de memoria donde se guardarán los datos recibidos
    - **length:** número de bytes a recibir

IPC en Unix: sockets

- **Llamada al sistema *write***: Transferencia de datos
  - write - write from socket
    - #include <unistd.h>
    - int write(int sockfd, char *buffer, int length);
  - Devuelve el número de bytes enviados por el socket
  - Argumentos de la llamada
    - **sockfd:** descriptor asociado al socket
    - **buffer:** @ de memoria donde se encuentra los datos a enviar
    - **length:** número de bytes a enviar

IPC en Unix: sockets

- **Llamada al sistema *send***: Transferencia de datos
  - send, - send a message from a socket
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - ssize_t send(int sockfd, char *message, int length, int flags);
  - Devuelve el número de bytes enviados por el socket
  - Argumentos de la llamada
    - **sockfd**
    - **message**
    - **length**
    - **flags**

IPC en Unix: sockets

- **Llamada al sistema *send***: Transferencia de datos
  - Argumentos de la llamada
    - **sockfd:** descriptor asociado al socket
    - **message:** @ de memoria donde se encuentra los datos a enviar
    - **length:** número de bytes a enviar
    - **flags:** bandera de control de la transmisión
      - **MSG_OOB** para enviar o recibir datos out-of-band
        - datos urgentes
      - **MSG_DONTROUTE** para enviar o recibir sin utilizar las tablas de routing
        - administradores de red y procesos privilegiados

IPC en Unix: sockets

- **Llamada al sistema *recv***: Transferencia de datos
  - recv - receive a message from a socket
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - #include <sys/uio.h>
    - int recv(int sockfd, char *buffer, int length, int flags);
  - Devuelve el número de bytes recibidos por el socket
  - Argumentos de la llamada
    - **sockfd**
    - **buffer**
    - **length**
    - **flags**

IPC en Unix: sockets

- **Llamada al sistema *recv***: Transferencia de datos
  - Argumentos de la llamada
    - **sockfd:** descriptor asociado al socket
    - **buffer:** @ de memoria donde se guardarán los datos recibidos
    - **length:** número de bytes a recibir
    - **flags:** bandera de control de la transmisión
      - **MSG_OOB** para enviar o recibir datos out-of-band
        - datos urgentes
      - **MSG_DONTROUTE** para enviar o recibir sin utilizar las tablas de routing
        - administradores de red y procesos privilegiados

IPC en Unix: sockets

- **Llamada al sistema *sendto***: Transferencia de datos
  - sendto - send a message from a socket
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - int sendto(int sockfd, char *message, int length, int flags, struct sockaddr *destaddr, int addrlen);
  - Devuelve el número de bytes enviados por el socket
  - Argumentos de la llamada
    - **sockfd, message, length**
    - **flags**
    - **destaddr, addrlen**

IPC en Unix: sockets

- **Llamada al sistema *sendto***: Transferencia de datos
  - Argumentos de la llamada
    - **sockfd:** descriptor asociado al socket
    - **message:** @ de memoria donde se encuentra los datos a enviar
    - **length:** número de bytes a enviar
    - **flags:** bandera de control de la transmisión
      - **MSG_OOB** para enviar o recibir datos out-of-band
        - datos urgentes
      - **MSG_DONTROUTE** para enviar o recibir sin utilizar las tablas de routing
        - administradores de red y procesos privilegiados
    - **destaddr**: especifica la dirección destino
    - **addrlen:** es la longitud en bytes de la dirección destino

IPC en Unix: sockets

- **Llamada al sistema *recvfrom***: Transferencia de datos
  - recvfrom - receive a message from a socket
    - #include <sys/types.h>
    - #include <sys/socket.h>
    - #include <sys/uio.h>
    - int recvfrom(int sockfd, char *message, int length, int flags, struct sockaddr *fromaddr, int addrlen);
  - Devuelve el número de bytes recibidos por el socket
  - Argumentos de la llamada
    - **sockfd, message, lenght**
    - **flags**
    - **fromaddr, addrlen**

IPC en Unix: sockets

- **Llamada al sistema *recvfrom***: Transferencia de datos
  - Argumentos de la llamada
    - **sockfd:** descriptor asociado al socket
    - **message:** @ de memoria donde se guardan los datos recibidos
    - **length:** número de bytes a enviar
    - **flags:** bandera de control de la transmisión
      - **MSG_OOB** para enviar o recibir datos out-of-band
        - datos urgentes
      - **MSG_DONTROUTE** para enviar o recibir sin utilizar las tablas de routing
        - administradores de red y procesos privilegiados
    - **destaddr**: se actualiza con la dirección de envío
    - **addrlen:** es la longitud en bytes de la dirección de envío

IPC en Unix: sockets

- **Llamada al sistema *close***: Cerrar el socket
  - close – close a socket
    - #include <unistd.h>
    - int close(int sockfd);
  - Notar que TCP asegurará que antes de destruir el socket todos los datos que se han enviado serán recibidos
    - El usuario puede pedir que **NO** se haga esta comprobación final mediante la llamada **shutdown( )**

IPC en Unix: sockets

- **Llamada al sistema *shutdow***: Cerrar el socket
  - shutdown - shut down part of a full-duplex connection
    - int shutdown(int sockfd, int howto);
  - Argumentos de la llamada
    - **sockfd:** descriptor asociado al socket
    - **howto:**
      - = 0 indica que los datos que se puedan recibir ya no interesan
      - = 1 indica que los datos que aún no se han enviado no se envíen
      - = 2 indica que el socket se cierre de inmediato haya lo que haya pendiente, enviar datos o recibir datos

IPC en Unix: sockets

- **Obtener información de la máquina:** ¿Cómo obtener direcciones IP?
  - Mediante funciones de librería que proporcionan información sobre una máquina dada o sobre su dirección IP
  - Estas funciones consultan ficheros o bases de datos internas
  - gethostname - get name of current host
    - #include <unistd.h>
    - int gethostname(char *hostname, int length);
  - Devuelve 0 si todo va bien y -1 en caso de error
  - Proporciona el nombre de un host

IPC en Unix: sockets

- **Obtener información de la máquina:** ¿Cómo obtener direcciones IP?
  - gethostbyname - get network host entry
    - #include <netdb.h>
    - struct hostent *gethostbyname(const char *name);
  - Devuelve un puntero a la estructura **hostent**
  - Argumentos de la llamada
    - **name:** nombre del host

IPC en Unix: sockets

- **Obtener información de la máquina:** ¿Cómo obtener direcciones IP?
  - gethostbyaddr - get network host entry
    - #include <netdb.h>
    - struct hostent *gethostbyaddr(const char *addr, int len, int type);
  - Devuelve un puntero a la estructura **hostent**
  - Argumentos de la llamada
    - **addr:** secuencia de bytes que contiene la dirección IP del host
    - **len,** tamaño en bytes de addr
    - **type,** tipo de addr

IPC en Unix: sockets

- **Obtener información de la máquina:** ¿Cómo obtener direcciones IP?
  - La estructura **hostent** está definida en <netdb.h>
  - La estructura **hostent** contiene el nombre y la dirección IP del host
  - struct hostent

```
{
    char  *h_name;              /* nombre oficial del host  */
    char  **h_aliases;          /* lista de alias  */
    int  h_addrtype;       /* tipo de direcciones de hosts (AF_INET) */
    int  h_length;              /*  longitud de la dirección  */
    char  **h_addr_list;     /* lista de direcciones, acaba en NULL  */
    #define h_addr  h_addr_list[0]
};
```


IPC en Unix: sockets

- **Obtener información sobre la red**
  - Existen funciones de librería que permiten obtener información sobre la red
  - Fichero: /etc/networks
  - getnetbyname - get network entry
    - #include <netdb.h>
    - struct netent *getnetbyname(const char *name);
  - Devuelve un puntero a la estructura **netent**
  - Argumentos de la llamada
    - **name:** nombre de la red

IPC en Unix: sockets

- **Obtener información sobre la red**
  - getnetbyaddr - get network entry
    - #include <netdb.h>
    - struct netent *getnetbyaddr(long net, int type);
  - Devuelve un puntero a la estructura **netent**
  - Argumentos de la llamada
    - **net:** es la dirección de la red en 32 bits
    - **type:** es un entero que especifica el tipo de red

IPC en Unix: sockets

- **Obtener información sobre la red**
  - La estructura **netent** está definido en <netdb.h>
  - La estructura **netent** contiene el nombre y el número de la red
  - struct netent

```
{
    char  *n_name;      /*  nombre oficial de la red      */
    char  **n_aliases;  /*  lista de alias                */
    int   n_addrtype;   /*  tipo de la dirección de la red */
    int   n_net;        /*  número de la red              */
};
```


IPC en Unix: sockets

- **Obtener información sobre los servicio de red**
  - Existen números de puertos, ports, reservados para servicios como FTP o TELNET
  - Existen funciones de librería para obtener información sobre los servicios y los puertos que utilizan
  - Estos servicio están definidos en /etc/services (**)

IPC en Unix: sockets

- **Obtener información sobre los servicio de red**
  - getservbyname - get service entry
    - #include <netdb.h>
    - struct servent *getservbyname(const char *name, const char *proto);
  - Devuelve un puntero a la estructura **servent**
  - Argumentos de la llamada
    - **name:** nombre del servicio del que desea obtener información
    - **proto:** protocolo que utiliza el servicio

IPC en Unix: sockets

- **Obtener información sobre los servicio de red**
  - getservbyport - get service entry
    - #include <netdb.h>
    - struct servent *getservbyport(int port, const char *proto);
  - Devuelve un puntero a la estructura **servent**
  - Argumentos de la llamada
    - **port:** puerto del servicio
    - **proto:** protocolo que utiliza el servicio

IPC en Unix: sockets

- **Obtener información sobre los servicio de red**
  - La estructura **servent** está definido en <netdb.h>
  - La estructura **servent** contiene el nombre de un servicio y el puerto que lo soporta
  - struct servent

```
{
    char  *s_name;        /*  nombre oficial del servicio  */
    char  **s_aliases;    /*  lista de alias               */
    int  s_port;          /*  numero de puerto             */
    char  *s_proto;       /*  protocolo a usar             */
};
```


IPC en Unix: sockets

- **Obtener información de los protocolos**
  - Existen funciones de librería que permiten obtener información sobre los protocolos disponibles en la máquina
  - Cada protocolo posee un nombre oficial, un alias registrado y un número oficial
  - Están definidos en /etc/protocols (**)
  - getprotobyname - get protocol entry
    - #include <netdb.h>
    - struct protoent *getprotobyname(const char *name);
  - Devuelve un puntero a la estructura **protoent**
  - Argumentos de la llamada
    - **name**: nombre del protocolo del que se desea información

IPC en Unix: sockets

- **Obtener información de los protocolos**
  - getprotobynumber - get protocol entry
    - #include <netdb.h>
    - struct protoent *getprotobynumber(int proto);
  - Devuelve un puntero a la estructura **protoent**
  - Argumentos de la llamada
    - **proto**: número del protocolo del que se desea información

IPC en Unix: sockets

- **Obtener información de los protocolos**
  - La estructura **protoent** está definido en <netdb.h>
  - struct protoent
    {
    char *p_name; /* nombre oficial del protocolo */
    char **p_aliases; /* lista de alias */
    int p_proto; /* número de protocolo */
    };

IPC en Unix: sockets

- **Funciones de conversión**
  - ¿Cómo solucionar los problemas generados de los diferentes tipos de almacenamiento interno de los datos?
    - big-endian
    - little-endian
  - El protocolo de red especifica el orden de transferencia
  - Todas las máquinas que lean o escriban enteros a través de esa red escribirán sus enteros en el formato de red

IPC en Unix: sockets

- **Funciones de conversión**
- htonl, htons, ntohl, ntohs - convert values between host and network byte order
  - #include <sys/types.h>
  - #include <netinet/in.h>
  - #include <inttypes.h>
  - u_long htonl(u_long hostlong); /* host to network long */
  - u_long ntohl(u_long netlong); /* network to host long */
  - u_short htons(u_short hostshort); /* host to nerwork short */
  - u_short ntohs(u_short netshort); /* network to host short */
  - **htonl( ):** convierte al formato estándar de red, enteros de 32 bits
  - **ntohl( ):** es el inverso de **htonl( )**
  - **htons( ):** convierte al formato estándar de red, enteros de 16 bits
  - **ntohs( ):** inverso a **htons( )**

IPC en Unix: sockets

- **Funciones especiales para manipular cadenas**
  - Existen funciones de C para tratar cadenas de caracteres que no terminal con el byte '\0', NULL
  - bstring, bcopy, bcmp, bzero - bit and byte string operations
    - #include <strings.h>
    - void bcopy(char *src, char *dest, int nbytes);
    - void bzero(char *dest, int nbytes);
    - int bcmp(char *ptr1, cahr *ptr2, int nbytes);
  - **bcopy:** copia nbytes de la cadena fuente a la destino
  - **bzero:** escribe nbytes NULL en la cadena destino
  - **bcmp:** compara dos cadenas de bytes arbitrarias devolviendo 0 si son idénticas

IPC en Unix: sockets

- **Funciones para la manipulación de direcciones IP**
  - ined_addr - Internet address manipulation
    - #include <arpa/inet.h>
    - unsigned long inet_addr(char *ptr);
  - Convierte una cadena, en notación decimal, en una dirección de Internet de 32 bits
  - inet_ntoa - Internet address manipulation
    - #include <arpa/inet.h>
    - char *inet_ntoa(struct in_addr inaddr);
  - Convierte una dirección de Internet de 32 bits en una cadena de caracteres

IPC en Unix: RPC

- Introducción a las RPC
- El proceso cliente en RPC
- El proceso servidor en RPC
- Aspectos de diseño en RPC
- Utilización del paquete RPC de Sun Microsystems
- Pasos para convertir llamadas locales en remotas
- Comandos asociados a la utilidad **rpcgen**

IPC en Unix: RPC

- **Introducción: RPC** - Remote Procedure Call
  - Las RPC constituyen el núcleo de muchos S. O. Distribuidos, son una mezcla entre llamadas a procedimientos y el paso de mensajes
  - Evolución:
    - RPC
    - Invocación de métodos remotos en CORBA
    - RMI en Java
  - Notar que utilizando RPC, el programador no necesita preocuparse de cómo se realiza la comunicación entre los procesos: IPC

IPC en Unix: RPC

- **Introducción:** Funcionamiento de las **RPC**
  - Un proceso, cliente, para ejecutar una RPC, envía un mensaje a otro proceso, servidor. Después, esperará el resultado
  - El mensaje contiene el nombre del procedimiento remoto a ejecutar y los argumentos
  - El proceso servidor, ejecutará el procedimiento remoto, extraerá los argumentos del mensaje, realizará la llamada al procedimiento de forma local, obtendrá los resultados y los enviará en otro mensaje al proceso cliente

IPC en Unix: RPC

- **Introducción:** Notas
  - **Llamadas al sistema**
    - Se ejecutan en un hilo de ejecución, thread, diferente al del programa que las invoca
  - **Funciones ordinarias**
    - Se ejecutan en el mismo hilo de ejecución, thread, del programa que las invoca
  - **RPC: Llamadas a procedimientos remotos**
    - Se ejecutan en un hilo de ejecución, thread, que pertenece al espacio de direcciones de otro proceso, servidor remoto

IPC en Unix: RPC

- **"El proceso cliente en RPC"**
  - Debe localizar al proceso, servidor, que ejecutará el procedimiento remoto
  - Debe construir un mensaje formado por:
    - procedimiento a ejecutar
    - parámetros
  - Debe enviar el mensaje al proceso servidor y esperar un mensaje de éste con la respuesta
  - El proceso cliente se bloqueará a la espera de la respuesta del proceso servidor, extrayendo posteriormente los resultados del mensaje enviado por éste

IPC en Unix: RPC

- **"El proceso servidor en RPC"**
  - El proceso servidor se encontrará en un bucle esperando la llegada de peticiones de los procesos clientes
  - Extraerá del mensaje los argumentos y el nombre del procedimiento que debe ejecutar
  - Realizará la llamada al procedimiento, que ya es un procedimiento propio del servidor
  - Con los resultados obtenidos, construirá un nuevo mensaje y se lo enviará al proceso cliente

IPC en Unix: RPC

- **Aspectos de diseño en RPC**
  - Las principales cuestiones de diseño a tener en cuenta para la implementación de un paquete RPC son:
    - El lenguaje de definición de interfaces: IDL
    - El proceso de generación de una RPC
    - La transferencia de los parámetros
    - El enlace dinámico
    - La semántica de las RPC en presencia de fallos

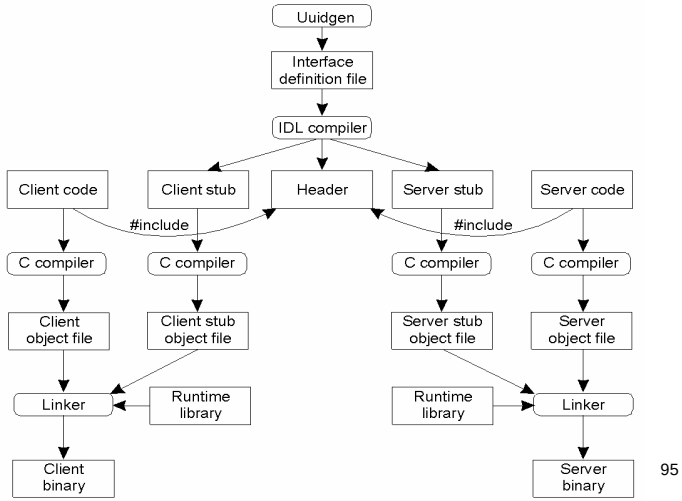
IPC en Unix: RPC

- **Diseño de RPC:** IDL, el lenguaje de definición de interfaces
  - Un interfaz especifica el nombre de un servicio que utilizarán clientes y servidores, por tanto incluye:
    - El nombre de los procedimientos
    - Parámetros de entrada
    - Parámetros de salida
  - La base de una aplicación cliente/servidor utilizando RPC está en:
    - Diseñar un interfaz
    - Escribir el interfaz en algún IDL: Interfaz Definition Language

IPC en Unix: RPC

- **Diseño de RPC:** IDL, el lenguaje de definición de interfaces
  - Notar que a partir de una interfaz escrita en algún IDL, el compilador de interfaces se encargará de obtener los resguardos del cliente y del servidor:
    - client stube
    - server stube

RPC: Proceso de generación de una RPC



IPC en Unix: RPC

- **Diseño de RPC:** IDL, el lenguaje de definición de interfaces
  - Los resguardos del cliente y del servidor se generan de forma automática
  - Los resguardos en el cliente se encargan de:
    - Localizar el servidor
    - Empaquetar los parámetros y construir los mensajes que se envían al servidor
    - Esperar la recepción del mensaje del servidor con la respuesta
    - Desempaquetar los resultados del mensaje de respuesta

IPC en Unix: RPC

- **Diseño de RPC:** IDL, el lenguaje de definición de interfaces
  - Los resguardos en el servidor realizan las tareas:
    - Localizar el cliente
    - Desempaquetar los parámetros enviados por el cliente
    - Empaquetar los resultados del mensaje de respuesta
  - Notar que los resguardos del cliente y del servidor son independientes de sus implementaciones, sólo dependen del interfaz

IPC en Unix: RPC

- **Diseño RPC:** La transferencia de los parámetros
  - ¿Qué problemas aparecen al empaquetar y desempaquetar los datos en las transferencias de los mensajes?
  - La representación de los datos
    - La representación de los datos puede ser distinta en las máquinas del cliente y del servidor: big-endian & little-endian
  - Solución:
    - Utilizar una representación intermedia de los datos utilizando un estándar
    - Ejemplo.- **XDR**: eXternal Data Representation

IPC en Unix: RPC

- **Diseño RPC:** El enlace dinámico
  - Permite la localización de los objetos con nombre en un S. D., es decir, localiza los servidores que ejecutan las RPC
  - Fases:
    - 1ª El servidor registra en el servidor de nombres, o enlazador dinámico, los nombres de los procedimientos que exporta junto con su dirección
      - Ejemplo.- Utilizando TCP/IP registraría la dirección IP y el puerto en el que se encuentra escuchando
    - 2ª Cuando el cliente ejecuta una RPC busca en el servidor de nombres la dirección del servidor que exporta un determinado servicio
    - 3ª El servidor de nombres envía al cliente la dirección del proceso servidor que exporta un determinado servicio

IPC en Unix: RPC

- **Diseño RPC:** El enlace dinámico
  - Dos tipos de enlace:
    - Enlace NO persistente
      - La conexión entre el cliente y el servidor se establece cada vez que se ejecuta una RPC
      - Es ineficiente cuando cliente ejecute muchas RPC de forma repetitiva
      - **Ventaja**: los servidores pueden migrar de un procesador a otro y los clientes no se ven afectados
    - Enlace persistente
      - La conexión entre el cliente y el servidor se mantiene después de la primera RPC
      - Útil cuando las aplicaciones ejecutan muchas RPC repetidas
      - Presenta problemas cuando los servidores cambian de nodo

IPC en Unix: RPC

- **Diseño RPC:** El enlace dinámico
  - Notar que la localización de los servicios que ofrecen los servidores es una de las tareas del resguardo del cliente
  - Notar que existen servicios que permiten dar de baja un procedimiento en el servidor de nombres

IPC en Unix: RPC

- **Diseño RPC:** Semántica de las RPC en presencia de fallos
  - Problemas que presentan las RPC:
    - El cliente no es capaz de localizar al servidor
    - Se pierde el mensaje de solicitud del cliente
    - Se pierde el mensaje de respuesta del servidor
    - El servidor falla después de recibir una petición
    - El cliente falla después de enviar una petición
  - Notar que la semántica de las RPC determina que ocurre cuando se repite una RPC
    - Def.- Una operación es ídem potente si se puede repetir tantas veces como sea necesario sin ocasionar ningún problema
      - Las transferencias bancarias NO son operaciones ídem potentes

IPC en Unix: RPC

- **Diseño RPC:** Semántica de las RPC en presencia de fallos
  - Tipos de semántica en presencia de fallos de las RPC:
    - **Al menos una vez:** garantizan con reintentos que la RPC se ejecuta al menos una vez
    - **A lo más una vez:** no hay reintentos, pude que no se llega a realizar la RPC ni una sola vez
    - **Exactamente una vez:** Se corresponde con las llamadas a procedimientos convencionales. Su implementación es difícil en un S. O. Distribuido

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems.**
  - Se diseñó para el desarrollo del sistema de ficheros en red NFS: Network File System
  - Los formatos de datos que posee están descritos utilizando la representación XDR, que también es de Sun
  - Un programa hace referencia a un servicio
  - Cada programa consta de una serie de funciones o procedimientos que pueden ser utilizados por los clientes
  - Cada programa puede tener varias versiones
  - Los programas, versiones y procedimientos se identifican mediante números enteros
  - La terna < programa, versión, procedimiento > identifica cada procedimiento

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** Principales tipos de datos en XDR
  - Enteros con signo:
    - int n; — En **C**: int n;
  - Enteros sin signo:
    - unsigned n; — En **C**: u_int n;
  - Valores lógicos:
    - bool n;
  - Números en coma flotante:
    - float n; — En **C**: float n;
  - Cadenas de bytes de longitud fija:
    - opaque cadena[256]; — En **C**: char cadena[256]

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** Principales tipos de datos en XDR
  - Cadenas de bytes de longitud variable:
    - opaque cadena <256>; — opaque cadena < >;
  - Cadenas de caracteres:
    - string cadena1<256>; — En **C**: char *cadena1;
  - Vectores de tamaño fijo:
    - int vector[100]; — En **C**: int vector[100];
  - Vectores de tamaño variable:
    - int vector <100>;
    - float vecotor2 < >;

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** Principales tipos de datos en XDR:
  - Estructuras:

```
struct estructura
  {
        int c1;
        string cadena2;
  };
```

  - Constantes:
    - const MAX = 12; — En **C**: #define MAX 12

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** La utilidad rpcgen
  - La utilidad rpcgen genera; a partir de un fichero que contiene las especificaciones de una aplicación RPC, aplicacion.x; los siguientes ficheros:
  - **fichero .h**: aplicación**.h**
    - Contiene las estructuras de datos en C equivalentes a las definidas en el fichero de especificación
    - Este fichero deberá incluirse en el código del cliente y en el código del servidor
  - **fichero _xdr.c**: aplicación_xdr.c
    - Fichero con las funciones para transformar los datos a la representación XDR
    - Contiene una función para cada uno de los tipos de datos

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** La utilidad rpcgen
  - **fichero _clnt.c**: aplicación_clnt.c
    - Fichero con el resguardo del cliente
  - **fichero _svc.c**: aplicación_svc.c
    - Fichero con el resguardo del servidor

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** La utilidad rpcgen
  - rpcgen - an RPC protocol compiler

    rpcgen **infile**
  - Opciones:
    - -C indica que se utiliza ANSI C
    - -N permite el paso de más de un argumento
      - Los compiladores de interfaces actuales permiten que las RPC tengan múltiples argumentos
    - -a genera todos los ficheros de apoyo posibles
  - Notar que el fichero **infile** debe tener extensión **.x** para indicar que es un fichero de especificaciones de interfaces

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** La utilidad rpcgen
  - **Ejemplo:** $> rpcgen -C -a proto.x
  - **makefile.proto**
    - Fichero makefile para compilar y enlazar todos los ficheros fuente tanto del cliente como del servidor
  - **proto.h**
    - Fichero de cabecera que contiene todos los prototipos de las funciones generados a partir del fichero de especificación
  - **proto_xdr.c**
    - Fichero contienen las funciones de transformación XDR que los “stub” del cliente y del servidor necesitan

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** La utilidad rpcgen
  - **proto_clnt.c**
    - Fichero que contiene el "client stube" que no se modifica
  - **proto_svc.c**
    - Fichero que contiene el "server stube" que no se modifica
  - **proto_client.c**
    - Fichero que contiene el esqueleto del programa principal del cliente, este fichero contiene las llamadas ficticias a las RPC
  - **proto_server.c**
    - Fichero que los prototipos de los "stub" de las RPC que se deberán programar

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** El servidor
  - Los servidores deben contener la implementación de cada una de los procedimientos definidos en el fichero de definición de interfaces
  - tipo_resultado *procedimiento_V_svc (tipo_argumento arg, struct svc_req *sr);
  - Notar que al nombre del procedimiento se le añade el número de versión, “V” y el sufijo “svc”

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** El cliente
  - Para que un cliente pueda ejecutar un procedimiento remoto debe establecer una conexión con el servidor mediante la llamada:
    - CLIENTE *clnt_create (char *host, u_long prognum, u_long vers_num, char * protocol);
  - Argumentos de la llamada:
    - **host**: nombre de la máquina donde se ejecuta el servidor
    - **prognum**: número de programa del procedimiento remoto a ejecutar
    - **vers_num**: versión del procedimiento remoto a ejecutar
    - **protocol**: protocolo de transporte empleado: UDP / TCP

IPC en Unix: RPC

- **Utilización del paquete RPC de Sun Microsystems:** El cliente
  - El prototipo que debe emplear el cliente para las llamadas a procedimientos remotos es:
    - tipo_resultado *procedimiento_V (tipo_argumento arg, CLIENTE *cl);
    - V representa el número de versión
  - Notar que las RPC de Sun utilizan los números de versiones para permitir que existan servidores que ofrezcan los mismos servicios con interfaces distintas

IPC en Unix: RPC

- **Pasos para convertir llamadas locales en remotas**
  - Lograr que los programas funcionen utilizando funciones locales
  - Reestructurar las funciones para que sólo tengan un parámetro, que se pasará por valor, y asegurar su funcionamiento local
  - Crear un fichero de especificación con extensión .x
  - Llamar a la utilidad rpcgen con las opciones -a, -C, -N para generar el conjunto completo de ficheros de apoyo
  - Compilar todos los fuentes, utilizando el fichero makefile, para detectar posibles errores en las definiciones de los tipos de datos definidos en el fichero de especificación
  - Insertar el código asociado al programa invocador en el fichero **nombre_client.c** generado por la utilidad rpcgen

IPC en Unix: RPC

- **Pasos para convertir llamadas locales en remotas**
  - Insertar el código de las funciones locales en el fichero **nombre_server.c** generado por la utilidad rpcgen
  - Compilar los fuentes definitivos utilizando el fichero makefile
  - Notar que al permitir múltiples versiones del mismo servicio, éstos se puedan actualizar y sin que haya que modificar los clientes que utilizan versiones antiguas

IPC en Unix: RPC

- **Dos comandos interesantes**
  - **rpcbind**
    - Permite registra un servicio RPC que está escuchando en un puerto
  - **rpcinfo**
    - Informa sobre los servicios de RPC están disponibles en un nodo dado
    - El campo **netid** indica el proveedor de transporte subyacente, UDP o TCP
    - rpcinfo -d elimina un servicio

IPC en Unix: RPC

- **Ejemplo:**
  - Fichero de interfaz para la definición de llamadas a procedimientos remotos, RPC
  - **calcular.x**

```
program CALCULAR
   {
        version UNO
        {
                int sumar(int a, int b) = 1;
                int restar(int a, int b) = 2;
        } = 1;
   } = 99;
```