



75.60 Sistemas Distribuidos I *Curso 2004*

Ambientes de Programación Cliente-Servidor TCP/IP

Prof. María Feldgen

Ambiente de Programación SOCKETS

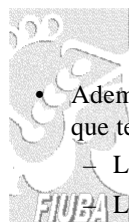
- **API (Application Programming Interface):** Interface entre el programa de aplicación y el software del protocolo.
- **IPC (Interprocess Communication):** provee comunicación entre procesos de un sistema o de sistemas diferentes.
- **API's existentes desarrolladas para lenguaje "C":**
 - **Sockets (4.3BSD)** Desarrollado por la Universidad de Berkeley, es la mas frecuente en Unix, Microsoft (Winsock), OS/2 y otros.
 - **TLI (Transport Layer Interface):** de Unix System V 3.0, desarrollado por AT&T.
- **TCP/IP** no provee ningún mecanismo para crear automáticamente un programa cuando arriba un mensaje, por lo tanto el programa que recibe el mensaje debe estar activo antes de efectuarse la primera comunicación.
- **Convenciones:**
 - **Cliente:** quien inicia la comunicación
 - **Server:** que espera por pedidos de comunicación.

Funcionalidad de la Interface



- **Operaciones conceptuales que soporta:**
 - Alocar recursos locales para la comunicación
 - Especificar las “puntas” local y remota de la comunicación
 - Iniciar una conexión (el cliente)
 - Esperar por una conexión entrante (el server)
 - Enviar o recibir datos
 - Terminar un conexión en forma graciosa
 - Tratar condiciones anormales de terminación
 - Desalocar recursos cuando la comunicación termina
- **Interface conceptual:**
 - no especifica representación de datos o detalles de programación.
 - es un conjunto de procedimiento y funciones. Sugiere los parámetros que requieren y la semántica de las operaciones
 - es “**débil**”, solo ilustra como las aplicaciones interactúan con TCP/IP, por lo cual las implementaciones difieren unas de otras.

UNIX I/O con TCP/IP



- Además del paradigma de **I/O de UNIX** (*open-read-write-close*) hay que tener en cuenta:
 - La relación cliente-servidor no es simétrica.
 - La comunicación puede ser con o sin conexión.
 - El protocolo de red es sin conexión
 - Se debe trabajar con los nombres de los hosts para autenticación
 - Se debe enviar la información en el **byte order** de la red
 - Se requieren parámetros específicos para la comunicación
 - El UNIX I/O es **stream oriented** y no **message oriented**.
 - La interface de red soporta múltiples protocolos de comunicación.

Funciones de sockets

- **socket** Crea un descriptor para usar en la comunicación
 - **connect **** Conecta con un server remoto (*Cliente*)
 - **read/readv** Recibe datos sobre una conexión
 - **write/writev** Escribe datos sobre una conexión
 - **close** Termina la comunicación y cierra el descriptor socket
 - **bind * **** Vincula una dirección IP y un port local con un socket
 - **listen *** Open pasivo y determina la cola de mensajes
 - **accept *** Acepta la próxima conexión entrante
 - **recv/recvmsg** Recibe el próximo datagrama entrante
 - **recvfrom** Recibe el próximo datagrama y recuerda su origen
 - **send/sendmsg** Envía un datagrama
 - **shutdown** Termina la conexión TCP en 1 o ambas direcciones
 - **select** Espera por condiciones de I/O
- (*) Función del server (**) Función del cliente

Comparación con otros paradigmas

Server

Función	Sockets	TLI	Mensajes	FIFO
alocar espacio		t_alloc ()		
crear endpoint	socket ()	t_open ()	msgget ()	mknode () open ()
vincular dir.	bind ()	t_bind ()		
especificar cola	listen ()			
esperar conexión	accept ()	t_listen ()		
obtener ID nuevo		t_open () t_bind () t_accept ()		

Comparación continuación				
Función	Sockets	TLI	Mensajes	FIFO
Cliente				
alocar espacio		t_alloc ()		
crear endpoint	socket ()	t-open ()	msgget ()	open ()
vincular dirección	bind ()	t-bind ()		
conectar al server	connect ()	t_connect ()		
Ambos				
transferir datos	read ()	read ()	msgrcv ()	read ()
	write ()	write ()	msgsnd ()	write ()
	recv ()	t_rcv ()		
	send ()	t_snd ()		
datagramas	recvfrom ()	t_rcvudata ()		
	sendto ()	t_sndudata ()		
terminar	close ()	t_close ()	msgctl ()	close ()
	shutdown ()	t_sndrel ()		unlink ()
		t_snddis ()		

75.60 Sistemas Distribuidos I (Prof. María Feldgen)
Laboratorio de Sistemas Distribuidos Heterogéneos - FIUBA (2004)

Socket Addresses

- Definición de la estructura en **<sys/socket.h>**:

```
struct sockaddr {  
    u_short  sa_family;    /* address family: AF_XXX */  
    char     sa_data[14];  /* protocol specific */  
};
```

Internet = AF_INET
- Específico de la familia de protocolo Internet **<netinet/in.h>**:

```
struct in_addr {  
    u_long   s_addr; /* 32 bit netid/hostid */  
};  
  
struct sockaddr_in {  
    short    sin_family; /* AF_INET */  
    u_short  sin_port;   /* 16 bit port number */  
    struct in_addr sin_addr; /* 32 bit netid/hostid */  
    char     sin_zero[8]; /* unused */  
};
```

Family
Port (2 bytes)
Netid / Hostid (4 Bytes)
No se usa

75.60 Sistemas Distribuidos I (Prof. María Feldgen)
Laboratorio de Sistemas Distribuidos Heterogéneos - FIUBA (2004)

8

Para todos los system calls

- `#include <sys/types.h>`
`#include <sys/socket.h>`

System Call: socket

- Crear un socket sin nombre dentro de un dominio de comunicación.
`int socket (int family, int type, int protocol);`

Family		Type	
AF_INET	Internet	SOCK_STREAM	stream
AF_UNIX	Unix Internal	SOCK_DGRAM	datagram
AF_NS	Xerox NS	SOCK_RAW	raw
AF_IMPLINK	IMP link layer	SOCK_SEQPACKET	sequenced packet
		SOCK_RDM	Reliably delivered message

Family	Type	protocolo
AF_INET	SOCK_STREAM	TCP
	SOCK_DGRAM	UDP
	SOCK_RAW	ICMP
		RAW

75.60 Sistemas Distribuidos I (Prof. María Feldgen)
Laboratorio de Sistemas Distribuidos Heterogéneos - FIUBA (2004)

9

System Call: bind

- Asigna una dirección local a un socket.
`int bind (int sockfd, struct sockaddr *myaddr, int addrlen);`
- Tiene tres usos:
 - El server registra su port “bien conocido”
 - Un cliente registra una dirección específica para él mismo.
 - Un cliente sin conexión necesita asegurarse que se le asigna esa dirección única, para que el server pueda contestar.

System Call: connect

- Establece una conexión con un socket remoto. (Cliente)
`int connect (int sockfd, struct sockaddr *servaddr, int addrlen);`
- Ambiente con conexión: Establece la conexión entre el sistema local y el remoto.
- Ambiente sin conexión: Sirve para que solo se reciban datagramas de la dirección guardada en *servaddr* con ese socket..

75.60 Sistemas Distribuidos I (Prof. María Feldgen)
Laboratorio de Sistemas Distribuidos Heterogéneos - FIUBA (2004)

10

System Call: listen

- Preparar un socket para aceptar conexiones entrantes (Server)
int listen (int sockfd, int backlog);

System Call: accept

- Esperar y Aceptar conexiones (Server)
Crea un nuevo descriptor de socket, (asume server concurrente).
- **int accept (int sockfd, struct sockaddr *peer, int addrlen);**

System Calls: close y shutdown

- Cierra un socket en ambas direcciones y dealocar recursos:
int close (int sockfd);
- Cierra un conexión en uno u otro sentido o en ambos:
int shutdown (int sockfd, int direction);
direction: 0: FREAD: cerrar la mitad de la conexión de lectura
1: FWRITE: cerrar la mitad de la conexión de escritura
2: FREAD/FWRITE: cerrar en ambas direcciones.

System Calls: write, writev y send

- Similares al *write* standard de entrada/salida. Envía datos al sistema remoto
- **Con sockets conectados:**
int send (int sockfd, char *buff, int nbytes, int flags);
int write (int sockfd, char *buff, int nbytes);
int writev (int sockfd, char *iovector, int nvector);
iovector: vector del tipo iovec, en cada elemento: pointer al bloque (32 bits)
y longitud del bloque (int de 32 bits).
nvector: cantidad de entradas que tiene iovector.
flags es cero o está formado por alguna de las siguientes ctes:
 - MSG_OOB: send o receive datos out-of-band.
 - MSG_DONTROUTE: El mensaje no requiere ruteo (send/sendto)



System Calls: read, readv y recv

- Similar al *read* standard de entrada/salida. Recibe datos del remoto.

Con sockets conectados:

int recv (int sockfd, char *buff, int nbytes, int flags);

int read (int sockfd, char *buff, int nbytes);

int readv (int sockfd, char *iovector, int nvector);

iovector: vector del tipo *iovec* donde cada elemento contiene: pointer al bloque (dirección de 32-bits) y longitud del bloque (int de 32 bits).

nvector: cantidad de entradas que tiene *iovector*.

flags es cero o está formado por alguna de las siguientes ctes:

- MSG_OOB: send o receive datos out-of-band.
- MSG_PEEK: Recibe una copia de los datos sin consumirlos

System Calls: sendto y sendmsg

- Similares al *write* standard de entrada/salida. Envía datos al sistema remoto

- **Con sockets NO conectados:**

int sendto (int sockfd, char *buff, int nbytes, int flags

struct sockaddr *to, int addrlen);

int sendmsg (int sockfd, struct messagestruc *mess, int flags);

mess: contiene puntero a la dirección del socket, longitud de la dirección, puntero a un vector *iovec* y cantidad de elementos, puntero a una lista de permisos y su longitud.

System Calls: recvfrom y recvmsg

- Similar al *read* standard de entrada/salida. Recibe datos del remoto.

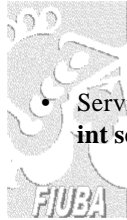
Con sockets NO conectados:

int recvfrom (int sockfd, char *buff, int nbytes, int flags

struct sockaddr *from, int addrlen);

int recvmsg (int sockfd, struct messagestruc *mess, int flags);

mess: contiene puntero a la dirección del socket, longitud de la dirección, puntero a un vector *iovec* y cantidad de elementos, puntero a una lista de permisos y su longitud.

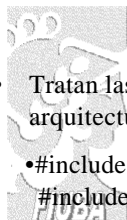


System Call: select

- Servidores que atienden servicios múltiples.

```
int select (int numfds, struct fd_set *refds,  
            struct fd_set *wrfds, struct fd_set *exfds,  
            struct timeval time);
```

- **numfds**: cantidad de descriptores de archivos en el conjunto.
- **refds**: Direcciones de los descriptores de archivos de entrada
- **wrfds**: Direcciones de los descriptores de archivos de salida
- **exfds**: Direcciones de los descriptores para excepciones.
- **time**: Máximo tiempo de espera o cero.



Ordenamiento de bytes

- Tratan las potenciales diferencias de ordenamiento de bytes entre diferentes arquitecturas de computadoras:

```
#include <sys/types.h>
```

```
#include <sys/netinet/in.h>
```

- Convertir **host-to-network**, long integer:

```
int htonl (u_long hostlong);
```

- Convertir **host-to-network**, short integer:

```
int htons (u_short hostshort);
```

- Convertir **network-to-host**, long integer:

```
int ntohl (u_long hostlong);
```

- Convertir **network-to-host**, short integer:

```
int ntohs (u_short hostshort);
```


Operaciones con bytes

- Mueve la cantidad de bytes que se especifican de origen a destino:
bcopy (char *src, char *dest, int nbytes);
- Escribe la cantidad de “null” bytes que se especifican:
bzero (char *dest, int nbytes);
- Compara dos string arbitrarios:
devuelve 0: si son iguales, caso contrario otro valor.
int bcmp (char *ptr1, char *ptr2, int nbytes);

Conversión de direcciones

- #include <sys/types.h>
#include <sys/netinet/in.h>
#include <arpa/inet.h>
- Convierte de formato “dotted decimal” a 32 bit Internet:
unsigned long inet_addr(char *ptr);
- Convierte una dirección 32 bit Internet en “dotted decimal”:
char *inet_ntoa(struct in_addr inaddr);

Lista de subrutinas de Información

- **Berkeley Network Library:**
 - **Para obtener información sobre hosts:**
 - **gethostbyname:** Obtiene la dirección dado el nombre del host
 - **gethostbyaddr:** Obtiene el nombre dada la dirección del host
 - **sethostent:** Usar una conexión TCP, para sucesivos queries.
 - **endhostent:** Cierra la conexión TCP de sethostent.
 - **Para obtener el nombre del host conectado:**
 - **getpeername:** Obtiene el nombre del “peer” conectado en el socket .
 - **Para obtener o cambiar el nombre del host local:**
 - **gethostname:** Obtiene el nombre del host local
 - **sethostname:** Cambia el nombre del host local
 - **Para obtener o cambiar el nombre del dominio:**
 - **getdomainname:** Obtiene el nombre del dominio del host
 - **setdomainname:** Cambia el nombre del dominio del host.
- **Otras implementaciones: (Ejemplo Linux):**
 - **Para obtener información del sistema del host local:**
 - **uname:** Devuelve el sistema operativo, hardware y dominio del host .

- **Berkeley Network Library:(continuación)**
 - **Obtener o cambiar la dirección IP del host local:**
 - **gethostid:** Obtiene la dirección IP del host local
 - **sethostid:** Cambia la dirección IP del host local
 - **Obtener o cambiar opciones de un socket:**
 - **getsockopt:** Obtiene
 - **setsockopt:** Cambia
 - **getsockname:** Obtiene el nombre de un socket
 - **Obtener información de /etc/protocols:**
 - **getprotoent:** Lee la siguiente línea de /etc/protocols
 - **getprotobyname:** Obtiene la inf. asociada al nombre
 - **getprotobynumber:** Obtiene la inf. asociada al número
 - **setprotoent:** Abre el archivo /etc/protocols
 - **endprotoent:** Cierra el archivo /etc/protocols
 - **Obtener información del archivo /etc/networks:**
 - **getnetent:** Lee la siguiente línea de /etc/networks
 - **getnetbyname:** Obtiene la inf. asociada al nombre de red
 - **getnetbynumber:** Obtiene la inf. asociada al número de red
 - **setnetent:** Abre el archivo /etc/networks
 - **endnetent:** Cierra el archivo /etc/networks

- **Berkeley Network Library (continuación):**
 - **Obtener información del archivo /etc/services:**
 - **getservent:** Lee la siguiente línea de /etc/services
 - **getservbyname:** Obtiene la inf. asociada al nombre
 - **getservbyport:** Obtiene la inf. asociada al port del servicio
 - **setservent:** Abre el archivo /etc/services
 - **endservent:** Cierra el archivo /etc/services
 - **Obtener información del archivo /etc/passwd:**
 - **getpwent:** Lee la siguiente línea de /etc/passwd
 - **getpwnam:** Obtiene la inf. asociada al "user name"
 - **getpwuid:** Obtiene la inf. asociada al "user uid"
 - **setpwent:** Abre el archivo /etc/passwd
 - **endpwent:** Cierra el archivo /etc/passwd
 - **fgetpwent:** idem getpwent
 - **putpwent:** graba una nueva entrada en /etc/passwd
 - **getpw:** Reconstruye una entrada en /etc/passwd

- **Berkeley Network Library (continuación):**
 - **Obtener la identidad del usuario local:**
 - **getuid:** Obtiene el ID del usuario real del proceso
 - **geteuid:** Obtiene el ID del usuario efectivo del proceso
 - **Obtener la identidad del grupo del usuario local:**
 - **getgid:** Obtiene el ID del grupo real del proceso
 - **getegid:** Obtiene el ID del grupo efectivo del proceso
 - **Obtener la identificación del proceso:**
 - **getpid:** Obtiene el *process ID* del proceso
 - **getppid:** Obtiene el *process ID* de padre del proceso actual
 - **Obtener o cambiar ID de procesos:**
 - **getpgid:** Obtiene el *process group ID* del proceso especificado
 - **setpgid:** Cambia el *process group ID* del proceso especificado
 - **getpgrp:** Obtiene el *process group ID* del proceso actual (equivalente a `getpgid(0)`)
 - **setpgrp:** Cambia el *process group ID* del proceso actual

- **Berkeley Network Library (continuación):**
 - **Obtener o cambiar la hora o “timezone” del host local:**
 - **gettimeofday:** Obtiene datos de la hora del host local
 - **settimeofday:** Cambia datos de la hora del host local
 - **Obtener o cambiar valores de un “interval timer” del host local:** hay 3 *interval timers*, *c/u* decrementándose en un *time domain* distinto. Cuando un timer expira, se envía una señal al proceso y el timer se reinicializa.
 - **getitimer:** Obtiene datos del timer indicado
 - **settimeofday:** Pone el timer en el valor indicado

Estructura hostent

- **struct hostent {**
 - char *h_name;** /* nombre oficial del host */
 - char **h_aliases;** /* listas de alias */
 - int h_addrtype;** /* tipo de dirección del host */
 - int h_length;** /* longitud de la dirección */
 - char **h_addr_list;** /* lista de direcciones name server */
 - /* (NULL termina la lista) */
- };**
- #define h_addr h_addr_list(0)** /* 1° dirección en la lista */

Subrutinas de información: detalle

- `#include <netdb.h>`
`extern int h_errno;`
- **gethostbyname**: devuelve un puntero a la estructura *hostent*, si fue exitoso o cero si hubo un error (codigo de error en la variable global *h_errno*).
`struct hostent *gethostbyname (const char *hostname);`
- **gethostbyaddr**:
`struct hostent *gethostbyaddr (const char *addr, int len, int type)`
addr: es un puntero a una estructura *in_addr*.
len: es el tamaño de la estructura
type: es **AF_INET** (actualmente)
devuelve un puntero a la estructura *hostent* si no hay error, caso contrario, devuelve cero (error en la variable global *h_errno*).
- **Errores (en *h_errno*)**:
 - **HOST_NOT_FOUND**: host desconocido
 - **NO_ADDRESS**: El nombre es válido, pero no tiene dir. IP.
 - **NO_RECOVERY**: Error no recuperable en el name server
 - **TRY_AGAIN**: Error temporario en un name server. Reintentar + tarde.

- **sethostent**: Si *stayopen* es verdadero (1), se debe usar un socket TCP de conexión para los sucesivos queries al name server. Sino los queries del name server usarán datagramas UDP.
`void sethostent (int stayopen);`
- **endhostent**:
`void endhostent(void);`
- **getpeername**:
`int getpeername (int s, struct sockaddr *name, int *namelen);`
 - *s* = socket
 - *namelen* = longitud en bytes del nombre
 - *name* = devuelve nombre del "peer" host conectado
- **resultado: exitoso**: devuelve cero
error: devuelve -1 y en *errno* el valor del error que puede ser:
 - **EBADF**: *s* no es un descriptor válido
 - **ENOTSOCK**: *s* no es un socket, es un archivo
 - **ENOTCONN**: el socket no está conectado
 - **ENOBUFS**: No hay recursos suficientes para la operación
 - **EFAULT**: El parámetro name es un pointer inválido.

- `#include <unistd.h>`
- **gethostname:**
`int gethostname (char *name, size_t len);`
- **sethostname:** (solo para superuser)
`int sethostname (const char *name, size_t len);`
 - *len* = longitud en bytes del nombre
 - *name* = devuelve/pone el nombre del host
- **getdomainname:**
`int getdomainname (char *name, size_t len);`
- **setdomainname:** (solo para superuser)
`int setdomainname (const char *name, size_t len);`
 - *len* = longitud en bytes del nombre
 - *name* = devuelve/pone el nombre del host
- **resultado en todas las funciones**
 - **exitoso:** devuelve cero
 - **error:** devuelve -1 y en *errno* el valor del error que puede ser:
 - **EINVAL:** parametro len inválido
 - **EPERM:** el usuario no es el superuser para sethostname
 - **EFAULT:** El parámetro name es un pointer inválido.

75.60 Sistemas Distribuidos I (Prof. María Feldgen)
Laboratorio de Sistemas Distribuidos Heterogéneos - FIUBA (2004)

25


- **uname:**
`#include <sys/utsname.h>`
`int uname (struct utsname*buf);`

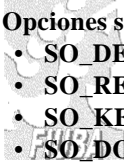
La estructura **utsname** está definida en */usr/include/sys/utsname.h*:

 - struct utsname {
 - char sysname[65];
 - char nodename[65];
 - char release[65];
 - char version[65];
 - char machine[65];
 - char domainname[65]; };
- **resultado: exitoso:** devuelve cero; **error:** -1 y en *errno* el error:
 - **EFAULT:** El parámetro buf es inválido.
- `#include <unistd.h>`
- **sethostid** (solo para superuser)
`int sethostid (long int hostid);`
 - *hostid* = dirección de IP, se almacena en el archivo */etc/hostid*
- **gethostid:** devuelve dirección de IP del host local, que puso *sethostid*.
`long int gethostid (void);`

75.60 Sistemas Distribuidos I (Prof. María Feldgen)
Laboratorio de Sistemas Distribuidos Heterogéneos - FIUBA (2004)

26

- 
- `#include <sys/types.h>`
`#include <sys/socket.h>`
 - **getsockopt:**
`int getsockopt(int s, int level, int opname, void*val, int len);`
 - **setsockopt:**
`int setsockopt(int s, int level, int opname, const void*val, int len);`
`level` = a nivel sockets es **SOL_SOCKET**
`val` = opciones que se reconocen a nivel sockets (ver tabla)
 - **resultado: exitoso:** devuelve cero
 - **error:** devuelve -1 y el error en **errno**:
 - **EBADF:** `s` no es un descriptor válido.
 - **ENOTSOCK:** `s` no es un socket, es un archivo
 - **ENONETOPT:** la opción es desconocida para ese nivel
 - **EFAULT:** El parámetro name es un pointer inválido.

- 
- **Opciones se reconocen a nivel sockets:**
 - **SO_DEBUG:** habilita que se registre información de debugging
 - **SO_REUSEADDR:** habilita que se reuse la dirección local
 - **SO_KEEPAIVE:** habilita que se haga un “keep alive” de la conexión
 - **SO_DONTROUTE:** bypass de ruteo para mensajes salientes.
 - **SO_BROADCAST:** permiso para transmitir mensajes broadcast.
 - **SO_LINGER:** bloquear el proceso hasta enviar todos los datos antes del close
 - **SO_OOBINDLINE:** recepción de “out-of-band data”.
 - **SO_SNDBUF / SO_RCVBUF:** definir tamaño del buffer de salida / entrada
 - **SO_SNDLOWAT / SO_RCVLOWAT:** contador mínimo salida/entrada
 - **SO_SNDTIMEO / SO_RCVTIMEO:** valor de timeout salida/entrada
 - **SO_TYPE:** obtener el tipo de socket (solo get)
 - **SO_ERROR:** obtener y borrar el error (solo get)

- `#include <sys/types.h>`
- `#include <sys/socket.h>`
- **getsockname:**
`int getsockname(int s, struct sockaddr *name, int namelen);`
- **resultado:exitoso:** devuelve cero
 - **error:** devuelve -1 y el error en *errno*:
 - **EBADF:** s no es un descriptor válido.
 - **ENOTSOCK:** s no es un socket, es un archivo
 - **ENOBUFS:** no hay suficientes recursos para hacer la operación
 - **EFAULT:** el parámetro name es un pointer inválido.

Subrutinas de Información (estructuras)

- `#include <netdb.h>`
- **La estructura *protoent*** definida en *netdb.h* es:

```
struct protoent { char *pname;          /* official protocol name */  
                  char **p_aliases;     /* alias list */  
                  int    p_proto;       /* protocol number */  
};
```
- **La estructura *netent*** definida en *netdb.h* es:

```
struct netent { char *n_name;           /* official network name */  
                char **n_aliases;      /* alias list */  
                int    n_addrtype;     /* net address type */  
                unsigned long int n_net; /* network number */  
};
```
- **La estructura *servent*** definida en *netdb.h*:

```
struct servent { char  *s_name;         /* nombre oficial del servicio */  
                  char  **s_aliases;    /* lista de aliases */  
                  int    s_port;        /* nro. de port (network byte order) */  
                  char  *s_proto;       /* protocolo a usar */  
};
```

- `#include <netdb.h>`
- **getprotobyname**: devuelve un puntero a la estructura *protoent* conteniendo la línea de */etc/protocols* que corresponde a *name*, si fue exitoso, o NULL si hubo error.
`struct protoent *getprotobyname (const char *name);`
- **getprotobynumber**: devuelve un puntero a la estructura *protoent* conteniendo la línea de */etc/protocols* que corresponde a *proto*, si fue exitoso, sino NULL.
`struct protoent *getprotobynumber (int proto);`
- **getprotoent**: devuelve un puntero a la estructura *protoent* conteniendo la línea siguiente de */etc/protocols*, si fue exitoso, o NULL si hubo un error.
`struct protoent *getprotoent (void);`
- **setprotoent**: Si *stayopen* es true (1), no se cierra el archivo */etc/protocols*, entre diferentes calls de *getprotobyname()* o *getprotobynumber()*.
`void setprotoent (int stayopen);`
- **endprotoent**:
`void endprotoent (void);`

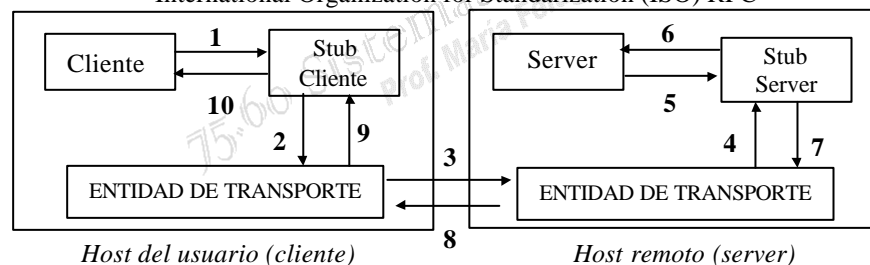
- **getnetbyname**: devuelve un puntero a la estructura *netent* conteniendo la línea de */etc/networks* que corresponde a *name*, si fue exitoso, o NULL si hubo error
`struct netent *getnetbyname (const char *name);`
- **getnetbynumber**: devuelve un puntero a la estructura *netent* conteniendo la línea de */etc/networks* que corresponde a *net*, si fue exitoso, sino NULL.
`struct netent *getnetbynumber (int net);`
- **getnetent**: devuelve un puntero a la estructura *netent* conteniendo la línea siguiente de */etc/networks*, si fue exitoso, o NULL si hubo un error.
`struct netent *getnetent (void);`
- **setnetent**: Si *stayopen* es true (1), no se cierra el archivo */etc/networks*, entre diferentes calls de *getnetbyname()* o *getnetbynumber()*.
`void setnetent (int stayopen);`
- **endnetent**:
`void endnetent (void);`

- `#include <netdb.h>`
- **getservbyname:**
`struct servent *getservbyname (char *servname, char *protname);`
- **getservbyport:**
`struct servent *getservbyport (int port, const char *protname);`
- **getservent:**
`struct servent *getservent (void);`
- Las anteriores devuelven un puntero a la estructura *servent*, o cero si hay error (*errno*).
- **setservent:** Si *stayopen* es true (1), no se cierra el archivo */etc/networks*, entre diferentes calls de *getservbyname()* o *getservbynumber()*.
`void setservent (int stayopen);`
- **endservent:**
`void endservent (void);`

Ambiente de Programación RPC

♦ RPC (Remote Procedure Call):

- Un proceso en una máquina local invoca a un procedimiento en una máquina remota.
- Distintas especificaciones e implementaciones:
 - Open Network Computing (ONC) RPC
 - Distributed Computing Environment (DCE) RPC
 - International Organization for Standardization (ISO) RPC



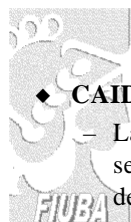


Requerimientos

- ◆ Pasaje de parámetros
- ◆ Binding
- ◆ Protocolo de transporte
- ◆ Tratamiento de excepciones
- ◆ Semántica del call
- ◆ Representación de datos
- ◆ Performance
- ◆ Seguridad

Implementación del protocolo

- ◆ Esquema *request - reply*
- ◆ Exclusión mutua en el server de procedimientos de un mismo programa.
- ◆ Sobre TCP o UDP manteniendo la semántica de la capa de transporte
- ◆ Estrategia simple de time out y retransmisión.



Semántica de RPC

◆ CAIDA DEL SERVER

- La operación entre cliente y server se puede repetir o no dependiendo del tipo de operación:
 - *Idempotente* (puede repetirse n veces)
 - *Exactamente una vez*
 - *A lo sumo una vez*
 - *Por lo menos una vez*

◆ CAIDA DEL CLIENTE:

- Tratamiento de huérfanos:
 - *Exterminio*
 - *Expiración*
 - *Reencarnación*
 - *Reencarnación gentil*

Binding

◆ Binding dinámico:

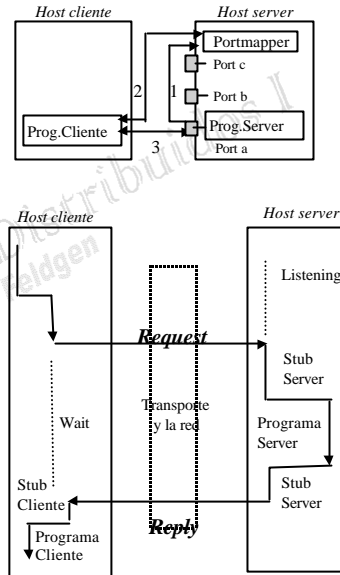
- El cliente localiza al server en forma dinámica, no se usan *ports* fijos.

◆ Especificación formal del servicio:

- Nombre del servicio
- Número de versión
- Lista de procedimientos provistos por ese servicio con sus parámetros indicando tipo: **in, out, in-out**

Interface del binder		Portmapper o Rpcbind	
Call	Input	Output	
Register	Nombre, Versión, Handle, Id. único		
Deregister	Nombre, Versión, Id.		
Lookup	Nombre, Versión	Handle, Id.	

75.60 Sistemas Distribuidos I (Prof. María Feldgen)
Laboratorio de Sistemas Distribuidos Heterogéneos - FIUBA (2004)



37

Standard ISO RPC

◆ Define un lenguaje y un protocolo RPC

◆ no define un método de implementación (*portabilidad a nivel código fuente*).

◆ El protocolo:

– **Semántica:** Al menos una vez

– **Esquema request - reply**

– Variantes:

- Broadcast:

– **Request** a múltiples servers

- Sin respuesta:

– **Request sin Reply**

– **Parámetros:** in, out e in-out

◆ Modelos:

- RPC Interaction Model

- RPC Communications Model

◆ Lenguaje: IDN

◆ Definición de servicios para:

- Basic RPC User Application-service-object (ASO),
- Signature Application-service-element (ASE),
- Basic RPC ASO y Basic RPC ASO-association Object.

◆ RPC utilizando ROSE y ACSE.

◆ Tipos de datos:

- menos tipos de datos que los otros
- exceptions

75.60 Sistemas Distribuidos I (Prof. María Feldgen)
Laboratorio de Sistemas Distribuidos Heterogéneos - FIUBA (2004)

38

DCE RPC

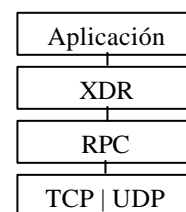
- ◆ OSF DCE (Open Systems Foundation, Distributed Computing Environment) RPC:
 - **Semántica:**
 - *por lo menos una vez e idempotente*
 - broadcast RPC y no-response RPC (*may be*)
 - soporta RPC autenticado
 - **Lenguaje:** IDL (Interface Definition Language)
 - **Parámetros:** in, out e in-out
 - **Tipos de datos:** del lenguaje “C” y handle_t (sin traducir) pipe (gran cantidad de datos)

ONC RPC (Sun RPC)

- ◆ El protocolo implementa:
 - **Semántica**
 - *A lo sumo una vez e idempotente.*
 - Esquema *request - reply*
 - Variantes: Broadcast y Sin respuesta (batching)
 - **Lenguaje:** rpcgen
 - **Parámetros:** in
 - **Tipos de datos:** del lenguaje “C” y opaque, union
 - **Autenticación:**
 - none (default)
 - Unix user ID/group ID
 - Secure RPC

Mecanismos de programación ONC RPC

- **Librerías XDR** (eXternal Data Representation)
- **Librerías Run Time:** para hacer:
 - llamadas (*call*) a procedimientos
 - registraci3n con el port mapper
 - despachar una llamada
- Un generador de programas (*rpcgen*).
- Ejecuci3n de los procedimientos del server:
 - Exclusi3n mutua o
 - Trabajar a nivel sockets con inetd.
- ◆ **Representaci3n de datos en XDR :**
 - Libreria para convertir:
 - datos individuales
 - datos complejos (estructuras) usados para definir mensajes RPC.



Stack de protocolos
en ambiente TCP/IP

