Sockets Básicos

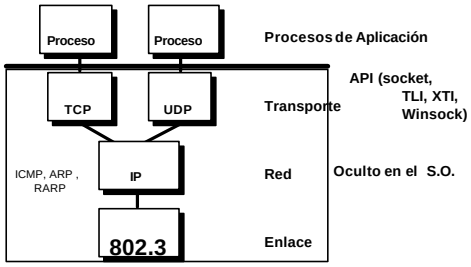
Pedro Merino Gómez
Jesus Martínez Cruz
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga

Sockets Básicos

Temas a tratar ...

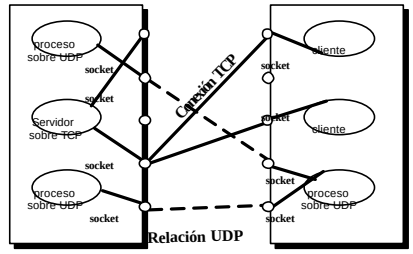
APIS para acceso a TCP/IP
El socket como descriptor para E/S
Puntos de conexión
Funciones básicas para acceso a TCP
Funciones básicas para acceso a UDP
Bases de datos de red

APIS para acceso a TCP/IP



APIS para acceso a TCP/IP

Los procesos acceden a TCP y UDP por Puertos

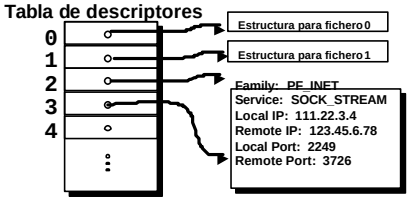


APIS para acceso a TCP/IP

Los Puertos se identifican por naturales
La relación de puertos con procesos se establece por el uso de funciones del S.O., entre ellas las de uso de sockets
Hay puertos de uso estándar para uso en Internet (ver /etc/services en UNIX)

El socket como descriptor para E/S

Un socket es un mecanismo de comunicación entre procesos del mismo o de sistemas diferentes
Se identifica por un descriptor similar al de ficheros



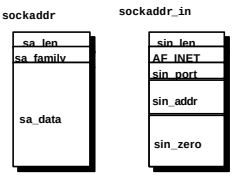
Puntos de conexión

El socket se asocia a un puerto y dirección IP locales
La estructura depende de la familia de protocolos

```
struct sockaddr
{
        u_short sin_family;
        char    sin_zero[14];
}
```

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

struct sockaddr_in
{
        u_short sin_family;
        u_short sin_port;
        u_long  sin_addr;
        char    sin_zero[];
}
```



Puntos de conexión

Todos los campos de sockaddr_in deben almacenarse en el orden en que se envían por la red

'**h**' : host byte order '**n**' : network byte order
'**s**' : short (16bit) '**l**' : long (32bit)

```
uint16_t htons(uint16_t);
uint16_t ntohs(uint_16_t);

uint32_t htonl(uint32_t);
uint32_t ntohl(uint32_t);
```

Puntos de conexión

```
struct  sockaddr_in sad; /* structure to hold an IP address     */
....


memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
sad.sin_family = AF_INET;         /* set family to Internet     */

if (argc > 2) {                   /* if protocol port specified   */
        port = atoi(argv[2]);     /* convert to binary            */
} else {
        port = PROTOPORT;         /* use default port number      */
      }
if (port > 0)                     /* test for legal value         */
  sad.sin_port = htons((u_short)port);
else {
      fprintf(stderr,"bad port number %s\n",argv[2]);
       exit(1);
}
```

Puntos de conexión

```
 /* Check host argument and assign host name. */

if (argc > 1) {
        host = argv[1];   /* if host argument specified   */
} else {
        host = localhost;
}

/* Convert host to equivalent IP address and copy to sad. */

ptrh = gethostbyname(host);
if ( ((char *)ptrh) == NULL ) {
        fprintf(stderr,"invalid host: %s\n", host);
        exit(1);
}
memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
```

Funciones básicas para TCP

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

family	PF_INET para IPv4
type	SOCK_STREAM, SOCK_DGRAM, SOCK_RAW.
proto	0

Crea la estructura para el descriptor

Funciones básicas para TCP

```
RETURN VALUES
     A -1 is returned if an error occurs.  Otherwise  the  return
     value is a descriptor referencing the socket.

ERRORS
     The socket() call fails if:

     EACCES
          Permission to create a socket of  the  specified  type
          and/or protocol is denied.

     EMFILE
          The per-process descriptor table is full.

     ENOMEM
          Insufficient user memory is available.

     ENOSR There were insufficient STREAMS resources available to
          complete the operation.

     EPROTONOSUPPORT
          The protocol type or the  specified  protocol  is  not
          supported within this domain.
```

VER MANUAL !!

Funciones básicas para TCP

```
int     sd;             /* socket descriptor
 */
……..
/* Create a socket. */

       sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
       if (sd < 0) {
              fprintf(stderr, "socket creation failed\n");
              exit(1);
       }
```

Funciones básicas para TCP

```
int  connect(int  s,
        const  struct  sockaddr   *name,
        int namelen);
```

name punto de conexión remoto

Conexión activa desde cliente
Ligadura automática al punto de conexión local

Funciones básicas para TCP

```
 int     sd;             /* socket descriptor                    */
……..
/* Connect the socket to the specified server. */

      if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
             fprintf(stderr,"connect failed\n");
             exit(1);
      }
```

Funciones básicas para TCP

```
int bind(int s,
             const struct sockaddr *name,
             int namelen);
```

name punto de conexión local !!!

Ligadura explícita al punto de conexión local

Funciones básicas para TCP

```
int     sd;             /* socket descriptor                    */
……..
memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
sad.sin_family = AF_INET;         /* set family to Internet     */
sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address   */
sad.sin_port = htons((u_short)port);

     /* Bind a local address to the socket */

if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
             fprintf(stderr,"bind failed\n");
             exit(1);
}
```

Funciones básicas para TCP

```
int listen(int s, int maxconec);
```

`maxconec` conexiones + peticiones pendientes

Esquema de cola de peticiones

Funciones básicas para TCP

```
int     sd;             /* socket descriptor                    */
...
 if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
              fprintf(stderr,"bind failed\n");
              exit(1);
      }

      /* Specify size of request queue */

      if (listen(sd, QLEN) < 0) {
              fprintf(stderr,"listen failed\n");
              exit(1);
      }
```

Funciones básicas para TCP

```
int accept(int sd,
        struct sockaddr *cli,
        int *clilen)
```

`cli` punto de conexión local al cliente

Termina la conexión TCP iniciada en el cliente
Devuelve:
- **Información sobre el cliente**
- **Un nuevo socket para emplear la conexión**

Funciones básicas para TCP

```
int     sd, sd2;            /* socket descriptor
 */
...
  while (1) {
          alen = sizeof(cad);
          if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0)
                      fprintf(stderr, "accept failed\n");
                      exit(1);
          }
          visits++;
          sprintf(buf,"This server has been contacted %d time%s\n",
                visits,visits==1?".":"s.");
                send(sd2,buf,strlen(buf),0);
                closesocket(sd2);
      }
```

Hay que conseguir atención a múltiples clientes !!

Funciones básicas para TCP

- Lectura
  - **`int read(int fd, char *buf, int max)`**

  Por defecto se bloquea hasta que haya datos disponibles.

  La lectura puede ser menor que el valor *max.* Los bytes reales leídos constituyen el valor de retorno de la llamada.

Funciones básicas para TCP

- Escritura
  - **`int write(int fd, char *buf, int max)`**

  La escritura puede ser menor que el valor *max.*

  Los bytes reales escritos constituyen el valor de retorno de la llamada.

Funciones básicas para TCP

- Cierre de la conexión
  - Cualquiera de los extremos puede efectuar la desconexión.
  - Llamada al sistema:
    - `close(int fd);`

Funciones para UDP

- Creación:
  - `socket( PF_INET, SOCK_DGRAM, 0)`
- **`Envío de un datagrama:`**
  - `sendto(int sockfd, void *buff, size_t nbytes, int flags, const struct sockaddr* to, socklen_t addrlen)`
- **`Recepción de un datagrama:`**
  - `recvfrom(int sockfd, void *buff, size_t nbytes, int flags, struct sockaddr* to, socklen_t *fromaddrlen)`

Funciones para UDP

- En envío:
  - Está permitido enviar 0 bytes.
  - El valor de retorno de la función indica el número de bytes que el S.O. **acepta mandar** como datagrama, no la cantidad de bytes que llegó al destino (es un servicio no fiable!).
  - **No** hay condición de error que indique que los datos no llegaron al destino!!

Funciones para UDP

- En recepción:
  - Está igualmente permitido recibir 0 bytes de datos.
  - El valor de retorno indica los bytes recibidos.
  - Si el buffer (`buff`) no es suficientemente grande, los datos que no caben **se pierden**.
  - `from` se rellena con los datos del emisor. Antes de la llamada, hay que especificar en `fromaddrlen` el tamaño de `from`.

Bases de datos de Red

- Existe una correspondencia entre nombres y direcciones IP:
  - 150.215.216.217 www.null.com [alias ...]
- Estos datos se encuentran en el fichero `/etc/hosts`
- Funciones:
  - gethostbyname
  - gethostbyaddr

Bases de datos de Red

gethostbyname

```
struct hostent * gethostbyname(const char* hostname)
```

- La estructura `hostent` está definida en `netdb.h`
  - Uno de sus campos es `h_addr`, que indica la dirección (en el orden de envío de la red) correspondiente a `hostname`. Es una cadena de caracteres!!
- Ante un error, devuelve NULL y da un valor a la variable global `h_errno`
- ¿Cómo se usa habitualmente?

```
h = gethostbyname("www.null.com");
memcpy(&sockaddr.sin_addr,h->h_addr,
  sizeof(struct in_addr));
```

Bases de datos de Red

gethostbyaddr

```
struct hostent *gethostbyaddr(const char *addr,
  size_t len,      /* sizeof(struct in_addr) */
  int family);     /* AF_INET */
```

- Aquí nos interesa el campo **`h_name`** de la estructura **`hostent`.**
- Ante un error, devuelve NULL y da un valor a la variable global **`h_errno`**

Bases de datos de Red

- Otras funciones de la librería:
  - **`uname`**: obtiene el nombre de la máquina
  - **`getservbyname`**: obtiene el puerto asignado a un servicio concreto, en **`/etc/services`**
  - **`getprotobyname`**: devuelve el identificador de un protocolo conocido por la máquina, en **`/etc/protocols`.**
  - **`getnetbyname`**: identifica la dirección asignada a una red, en **`/etc/networks`**