Programación de sockets

Los protocolos de la capa de aplicación pueden ser:

- **Estándar:** Se basan en una especificación, normalmente publicada en un RFC. Generalmente se les asocia números de puerto bien conocidos.
- **Propietarios:** No utilizan ninguna especificación pública, sino que están diseñados para una aplicación particular.

¿Cómo acceder a los servicios de la capa de transporte desde nuestras aplicaciones software?

Programación de sockets

Recuerda:

- Un **socket** es la combinación de la dirección IP + el puerto TCP o UDP

- **TCP**: Orientado a la conexión, confiable, y con control de flujo.
- **UDP**: Orientado a datagramas y no confiable.

La programación se puede hacer a través de distintas APIs y en distintos lenguajes de programación.

Programación de sockets

Un servidor sencillo con sockets UDP

Programación de sockets

```
int define_socket_UDP(int port) {
    struct sockaddr_in sin;
    int s;

    s = socket(AF_INET,SOCK_DGRAM, 0);

    if(s < 0) {
        errexit("No puedo crear el socket:
        %s\n", strerror(errno));
    }

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(port);

    if(bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        errexit("No puedo hacer el bind con el puerto: %s\n",
                strerror(errno));
    }

    return s;

}
```

La estructura sockaddr_in

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

Programación de sockets

```
int define_socket_UDP(int port) {
    struct sockaddr_in sin;
    int s;

    s = socket(AF_INET,SOCK_DGRAM, 0);

    if(s < 0) {
        errexit("No puedo crear el socket:
        %s\n", strerror(errno));
    }

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(port);

    if(bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        errexit("No puedo hacer el bind con el puerto: %s\n",
                strerror(errno));
    }

    return s;

}
```

La función socket

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

- **domain:** Indica la familia de procolos (AF_INET)
- **type:** El tipo de socket (SOCK_DGRAM ó SOCK_STREAM)
- **protocol:** Indica el protocolo que se utilizará con el socket. Generalmente lo dejamos 0.

Manual Linux: ip

Manual linux: socket

Programación de sockets

```
int define_socket_UDP(int port) {
    struct sockaddr_in sin;
    int s;

    s = socket(AF_INET,SOCK_DGRAM, 0);

    if(s < 0) {
        errexit("No puedo crear el socket:
        %s\n", strerror(errno));
    }

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(port);

    if(bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        errexit("No puedo hacer el bind con el puerto: %s\n",
                strerror(errno));
    }

    return s;

}
```

Rellenamos la estructura sockaddr

Los campos dependen de la familia de protocolos a utilizar

- **sin_family:** Indica la familia de procotolos, en este caso AF_INET.
- **sin_addr.s_addr:** Indica las direcciones que desde las cuales se aceptan conexiones.
- **sin_port:** Número de puerto.

Manual Linux: ip

Manual Linux: htons

Programación de sockets

```
int define_socket_UDP(int port) {
    struct sockaddr_in sin;
    int s;

    s = socket(AF_INET,SOCK_DGRAM, 0);

    if(s < 0) {
        errexit("No puedo crear el socket:
        %s\n", strerror(errno));
    }

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(port);

    if(bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        errexit("No puedo hacer el bind con el puerto: %s\n",
                strerror(errno));
    }

    return s;

}
```

Llamada a la función bind

```
int bind(
    int sockfd,
    const struct sockaddr *addr,
    socklen_t addrlen
);
```

- **sockfd:** Descriptor del socket.
- **addr:** Puntero a la estructura con la dirección.
- **addrlen:** Tamaño de la estructura sockadrr.

Manual Linux: bind

Manual Linux: ip

La función bind asocia la dirección y el puerto al socket.

Programación de sockets

```
int main(int argc, char **argv) {

  int port = 3300;
  char buf[1];
  time_t now;
  struct sockaddr_in fsin;
  socklen_t alen = sizeof(fsin);


  int s = define_socket_UDP(port);

  while (1) {

   if(recvfrom(s, buf, sizeof(buf), 0,
           (struct sockaddr *)&fsin, &alen) < 0)
           errexit("recvfrom: %s\n", strerror(errno));

   time(&now);
   now = htonl((u_long)now);

   sendto(s, &now, sizeof(mow), 0, (struct sockaddr *)&fsin, sizeof(fsin));
 }

}
```

Llamada a define_socket

Definimos el socket del que vamos a recibir los datos.

Programación de sockets

```
int main(int argc, char **argv) {

  int port = 3300;
  char buf[1];
  time_t now;
  struct sockaddr_in fsin;
  socklen_t alen = sizeof(fsin);


  int s = define_socket_UDP(port);

  while (1) {

   if(recvfrom(s, buf, sizeof(buf), 0,
            (struct sockaddr *)&fsin, &alen) < 0)
            errexit("recvfrom: %s\n", strerror(errno));

   time(&now);
   now = htonl((u_long)now);

   sendto(s, &now, sizeof(mow), 0, (struct sockaddr *)&fsin, sizeof(fsin));
 }
}
```

Llamada a recvfrom

```
ssize_t recvfrom(
    int sockfd,
    void *buf,
    size_t len,
    int flags,
    struct sockaddr *src_addr,
    socklen_t *addrlen
);
```

La función recvfrom bloquea el proceso hasta que se recibe un mensaje, salvo que el socket se haya definido como no bloqueante.

Manual: recvfrom

Programación de sockets

```
int main(int argc, char **argv) {

  int port = 3300;
  char buf[1];
  time_t now;
  struct sockaddr_in fsin;
  socklen_t alen = sizeof(fsin);


  int s = define_socket_UDP(port);

  while (1) {

   if(recvfrom(s, buf, sizeof(buf), 0,
            (struct sockaddr *)&fsin, &alen) < 0)
            errexit("recvfrom: %s\n", strerror(errno));

   time(&now);
   now = htonl((u_long)now);

   sendto(s, &now, sizeof(mow), 0, (struct sockaddr *)&fsin, sizeof(fsin));
 }

}
```

Llamada a sendto

```
ssize_t sendto(
    int sockfd,
    const void *buf,
    size_t len,
    int flags,
    const struct sockaddr *dest_addr,
    socklen_t addrlen
);
```

Manual Linux: htons
Manual: time
Manual: sendto

Programación de sockets

El cliente para el servidor sencillo con sockets UDP

Programación de sockets

```
int conectarUDP(char *host, int port) {
    struct sockaddr_in sin;
    struct hostent *hent;
    int s;

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);

    hent = gethostbyname(host);
    if(hent)
        memcpy(&sin.sin_addr, hent->h_addr, hent->h_length);
    else if ((sin.sin_addr.s_addr = inet_addr((char*)host)) == INADDR_NONE)
         errexit("No puedo resolver el nombre \"%s\"\n", host);


    s = socket(AF_INET, SOCK_DGRAM, 0);
    if(s < 0)
       errexit("No se puede crear el socket: %s\n", strerror(errno));

    if(connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
       errexit("No se puede conectar con %s: %s\n", host, strerror(errno));

    return s;

}
```

Rellenamos la estructura sockaddr con la familia de protocolos, dirección de host, y número de puerto

Manual Linux: ip

Programación de sockets

```
int conectarUDP(char *host, int port) {
    struct sockaddr_in sin;
    struct hostent *hent;
    int s;

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);

    hent = gethostbyname(host);
    if(hent)
        memcpy(&sin.sin_addr, hent->h_addr, hent->h_length);
    else if ((sin.sin_addr.s_addr = inet_addr((char*)host)) == INADDR_NONE)
         errexit("No puedo resolver el nombre \"%s\"\n", host);


    s = socket(AF_INET, SOCK_DGRAM, 0);
    if(s < 0)
       errexit("No se puede crear el socket: %s\n", strerror(errno));

    if(connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
       errexit("No se puede conectar con %s: %s\n", host, strerror(errno));

    return s;

}
```

Se utiliza la la función gethostbyname, para resolver el nombre del servidor.

Esta función es la interfaz con el servicio DNS

Manual: gethostbyname

Programación de sockets

```
int conectarUDP(char *host, int port) {
    struct sockaddr_in sin;
    struct hostent *hent;
    int s;

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);

    hent = gethostbyname(host);
    if(hent)
        memcpy(&sin.sin_addr, hent->h_addr, he          ength);
    else if ((sin.sin_addr.s_addr = inet_addr          r*)host)) == INADDR_NONE)
         errexit("No puedo resolver el nomb          %s\"\n", host);


    s = socket(AF_INET, SOCK_DGRAM, 0);
    if(s < 0)
       errexit("No se puede crear el socket: %s\n", strerror(errno));

    if(connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
       errexit("No se puede conectar con %s: %s\n", host, strerror(errno));

    return s;

}
```

Creamos un nuevo socket para la conexión.

Exactamente igual que en el lado del servidor

Manual Linux: ip

Manual linux: socket

Programación de sockets

```
int conectarUDP(char *host, int port) {
    struct sockaddr_in sin;
    struct hostent *hent;
    int s;

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);

    hent = gethostbyname(host);
    if(hent)
        memcpy(&sin.sin_addr, hent->h_addr, hen          gth);
    else if ((sin.sin_addr.s_addr = inet_addr((          ost)) == INADDR_NONE)
         errexit("No puedo resolver el nombre \          \n", host);


    s = socket(AF_INET, SOCK_DGRAM, 0);
    if(s < 0)
       errexit("No se puede crear el socket:  s\n", strerror(errno));

    if(connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
       errexit("No se puede conectar con %s: %s\n", host, strerror(errno));

    return s;
}
```

Llamada a la función connect

```
int connect(
    int sockfd,
    const struct sockaddr *addr,
    socklen_t addrlen
);
```

La función connect conecta el socket con una dirección remota y un puerto remoto.

Manual: connect

Programación de sockets

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

{

   char byte=0;
   int s = -1;
   time_t  now;
   char *host;
   int puerto;
   int n;

   host = argv[1];
   puerto = atoi(argv[2]);
   s = conectarUDP(host, puerto);

    send(s,&byte, 1,0);

    n = recv(s, (char *)&now, sizeof(now),0);
    if(n < 0)
        errexit("Error de lectura: %s\n", strerror(errno));

    now = ntohl((u_long)now);
    printf("%s", ctime(&now));

    exit(0);
}
```

Llamada a la función conectarUDP

Programación de sockets

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

{

   char byte=0;
   int s = -1;
   time_t  now;
   char *host;
   int puerto;
   int n;

   host = argv[1];
   puerto = atoi(argv[2]);
   s = conectarUDP(host, puert );

    send(s,&byte, 1,0);

    n = recv(s, (char *)&now, sizeof(now),0);
    if(n < 0)
        errexit("Error de lectura: %s\n", strerror(errno));

    now = ntohl((u_long)now);
    printf("%s", ctime(&now));

    exit(0);
}
```

Llamada a la función send

```
ssize_t send(
   int fd,
   const void *buf,
    size_t count,
   int flags
);
```

Manual: send

Programación de sockets

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

{

   char byte=0;
   int s = -1;
   time_t  now;
   char *host;
   int puerto;
   int n;

   host = argv[1];
   puerto = atoi(argv[2]);
   s = conectarUDP(host, puerto);

    send(s,&byte, 1,0);

    n = recv(s, (char *)&now, sizeof(now),0);
    if(n < 0)
        errexit("Error de lectura: %s\n", strerror(errno));

    now = ntohl((u_long)now);
    printf("%s", ctime(&now));

    exit(0);
}
```

Llamada a la función recv

```
ssize_t recv(
    int fd,
    const void *buf,
    size_t count,
    int flags
);
```

Manual: recv

Programación de sockets

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

{

   char byte=0;
   int s = -1;
   time_t  now;
   char *host;
   int puerto;
   int n;

   host = argv[1];
   puerto = atoi(argv[2]);
   s = conectarUDP(host, puerto);

    send(s,&byte, 1,0);

    n = recv(s, (char *)&now, sizeof(now),0
    if(n < 0)
        errexit("Error de lectura: %s\n", strerror(errno));

    now = ntohl((u_long)now);
    printf("%s", ctime(&now));

    exit(0);
}
```

Acciones de la aplicación

Manual Linux: htons

Programación de sockets

¿Cuál es el protocolo de aplicación que hemos implementado?

1) El cliente manda un byte para pedir la hora.
2) El servidor consulta la hora del sistema.
3) El servidor le responde con un paquete que contiene un entero con el número de segundos que han pasado desde las 0:00 del 1 enero 1970.
4) El cliente transforma este entero en una cadena de texto legible y aplica la configuración local sobre zonas horarias, y muestra la cadena por pantalla.

Programación de sockets

Un servidor sencillo con sockets TCP

Programación de sockets

```
int define_socket_TCP(int port) {
    struct sockaddr_in sin;
    int s;

    s = socket(AF_INET,SOCK_STREAM, 0);

    if(s < 0) {
        errexit("No puedo crear el socket:
        %s\n", strerror(errno));
    }

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(port);

    if(bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
        errexit("No puedo hacer el bind con el puerto: %s\n",
                strerror(errno));
    }

    if (listen(s, 5) < 0)
        errexit("Fallo en el listen: %s\n", strerror(errno));

    return s;

}
```

La única diferencia es el tipo de socket y la llamada a listen

Manual Linux: ip

Manual linux: socket

Programación de sockets

```
int define_socket_TCP(int port) {
    struct sockaddr_in sin;
    int s;

    s = socket(AF_INET,SOCK_STREAM, 0);

    if(s < 0) {
        errexit("No puedo crear el socket:
        %s\n", strerror(errno));
    }

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons(port);

    if(bind(s, (struct sockaddr *)&sin, sizeof(    ) < 0) {
        errexit("No puedo hacer el bind con el     rto: %s\n",
                strerror(errno));
    }

    if (listen(s, 5) < 0)
        errexit("Fallo en el listen: %s\n", strerror(errno));

    return s;

}
```

La función listen

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

Manual: listen

Programación de sockets

```
void cuenta_atras (int fd) {
   int i;
   for (i = 10; i >=0; i--) {
      int cuenta = htonl(i);
      send(fd,&cuenta, sizeof(int),0);
      sleep(1);
   }
}

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

{
    struct sockaddr_in fsin;
    int msock, ssock;
    int alen;
    msock = define_socket_TCP(3300);
    while (1) {
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
        if(ssock < 0)
            errexit("Fallo en el accept: %s\n", strerror(errno));

        cuenta_atras(ssock);
        close(ssock);
    }
}
```

La función de cuenta atrás recibe un descriptor de socket y escribe las cuentas en ese descriptor cada segundo.

Programación de sockets

```
void cuenta_atras (int fd) {
   int i;
   for (i = 10; i >=0; i--) {
      int cuenta = htonl(i);
      send(fd,&cuenta, sizeof(int),0);
      sleep(1);
   }
}

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

{
    struct sockaddr_in fsin;
    int msock, ssock;
    int alen;
    msock = define_socket_TCP(3300);
    while (1) {
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
        if(ssock < 0)
            errexit("Fallo en el accept: %s\n", strerror(errno));

    cuenta_atras(ssock);
        close(ssock);
    }
}
```

Llamada a la función define_socket_TCP

Programación de sockets

```
void cuenta_atras (int fd) {
   int i;
   for (i = 10; i >=0; i--) {
      int cuenta = htonl(i);
      send(fd,&cuenta, sizeof(int),0);
      sleep(1);
   }
}

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

{
    struct sockaddr_in fsin;
    int msock, ssock;
    int alen;
    msock = define_socket_TCP(3300);
    while (1) {
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
        if(ssock < 0)
            errexit("Fallo en el accept: %s\n", strerror(errno));

        cuenta_atras(ssock);
        close(ssock);
    }
}
```

La función accept

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

Manual: accept

La función accept extrae la primera conexión pendiente de la cola y devuelve un socket conectado.

Programación de sockets

```
void cuenta_atras (int fd) {
   int i;
   for (i = 10; i >=0; i--) {
      int cuenta = htonl(i);
      send(fd,&cuenta, sizeof(int),0);
      sleep(1);
   }
}

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

{
    struct sockaddr_in fsin;
    int msock, ssock;
    int alen;
    msock = define_socket_TCP(3300);
    while (1) {
        ssock = accept(msock, (struct so     ar *)&fsin, &alen);
        if(ssock < 0)
            errexit("Fallo en el    cept: %s\n", strerror(errno));

        cuenta_atras(ssock);
        close(ssock);
    }
}
```

La función close

int close(int fd)

Manual: close

Programación de sockets

Un cliente sencillo con sockets TCP

Programación de sockets

```
int conectarTCP(char *host, int port) {
    struct sockaddr_in sin;
    struct hostent *hent;
    int s;

    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);

    if(hent = gethostbyname(host))
        memcpy(&sin.sin_addr, hent->h_addr, hent->h_length);
    else if ((sin.sin_addr.s_addr = inet_addr((char*)host)) == INADDR_NONE)
        errexit("No puedo resolver el nombre \"%s\"\n", host);


    s = socket(AF_INET, SOCK_STREAM, 0);
    if(s < 0)
       errexit("No se puede crear el socket: %s\n", strerror(errno));

    if(connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
       errexit("No se puede conectar con %s: %s\n", host, strerror(errno));

    return s;

}
```

La única diferencia con la conexión en UDP es el tipo de SOCKET_STREAM

Programación de sockets

```
int main(int argc, char *argv[ ]) {
   int s = -1;
   int cuenta = -1;
   char *host;
   int puerto;
   int n;

   host = argv[1];
   puerto = atoi(argv[2]);
   s = conectarTCP(host, puerto);

   while(1)  {
        if (recv(s, (char*)&cuenta , sizeof(int),0)!=sizeof(int)) {
            errexit("Error de lectura\n");
        }
        int cuenta_reord = ntohl(cuenta);
        printf("\r%02d", cuenta_reord);
        fflush(stdout);

        if (cuenta_reord == 0) {
             break;
        }

    }

}
```

Conectamos al socket TCP remoto

Programación de sockets

```
int main(int argc, char *argv[ ]) {
   int s = -1;
   int cuenta = -1;
   char *host;
   int puerto;
   int n;

   host = argv[1];
   puerto = atoi(argv[2]);
   s = conectarTCP(host, puerto);

   while(1)  {
        if (recv(s, (char*)&cuenta , sizeof(int),0)!=sizeof(int)) {
            errexit("Error de lectura\n");
        }
        int cuenta_reord = ntohl(cuenta);
        printf("\r%02d", cuenta_reord);
        fflush(stdout);

        if (cuenta_reord == 0) {
             break;
        }

    }

}
```

Leemos las cuentas del socket

Programación de sockets

¿Cuál es el protocolo de aplicación que hemos implementado?

1) El cliente establece una conexión TCP con el servidor
2) El servidor envía las cuentas correspondientes cada segundo
3) Cuando una cuenta llega al cliente este la muestra por pantalla.

Programación de sockets

Implementación de servidores

Servidor sin conexión iterativo

```
Crear un socket

Hacer bind a un puerto bien
conocido

while (1)  {

  Leer petición del cliente

  Enviar respuesta al cliente

}
```

Servidor sin conexión concurrente

```
Crear un socket

Hacer bind a un puerto bien
conocido

while (1)  {

  Leer petición del cliente
  fork
  if (hijo)
    Enviar respuesta al cliente

}
```

Los servidores sin conexión utilizan el protocolo UDP

Programación de sockets

Implementación de servidores

Servidor con conexión concurrente

```
Crear un socket

Hacer bind a un puerto bien conocido
Llamada a listen

while (1)  {

  Aceptar una petición del cliente
  fork
  if (hijo) {
       Comunicar con el cliente
        Cerrar nuevo socket
  }
  else {
       Cerrar nuevo socket
  }

}
```

Programación de sockets

Más información en:

http://www.tenouk.com/cnlinuxsockettutorials.html

http://www.tenouk.com/download/pdf/Module40.pdf