Sockets

Roberto Gómez Cárdenas
rogomez@itesm.mx
http://webdia.cem.itesm.mx/ac/rogomez

o lo que es lo mismo… por que aprender sockets en C y no en Java

¿Qué falta de ver en sockets?

- Servidores concurrentes y procesos zombies
- Sockets y threads
- Sockets de tipo raw

Servidores concurrentes

```
sd = socket(AF_UNIX, SOCK_STREAM, 0 );
bind(sd, (struct sockaddr *)&saddr, addrlen );
listen(sd, 5);
while(1)  {
    fd = accept(sd, 0, 0 );
    if ( fork() = = 0 ) {          /*código proceso hijo */
       atencion_servicio(fd);
       exit(0);
    }
    else
        close(fd);                 /* padre no usa conexión */
}
```

Lámina 3 Roberto Gómez C.

Procesos zombies en servidores

- Proceso que no tiene recursos asignados, pero que ocupa un lugar en la tabla de procesos
- No es posible matarlo
- Dos formas de prevenirlos:
  - ignorando la señal (Sistema V)
  - atrapando la señal (BSD)

Lámina 4 Roberto Gómez C.

Ejemplo procesos zombies

```
rogomez@armagnac:25> ps -l
PID   PPID   TT     S     TIME  COMMAND
  :     :     :     :     :       :
  :     :     :     :     :       :
5525  3660  ----   IW    1:41    server
5527  5525  ----   Z     0:00    <defunct>
5529  5525  ----   Z     0:00    <defunct>
5531  5525  ----   Z     0:00    <defunct>
5537  36060 ----   S     0:16    tcsh
rogomez@armagnac:26>
```

Lámina 5 Roberto Gómez C.

Consecuencias procesos zombies

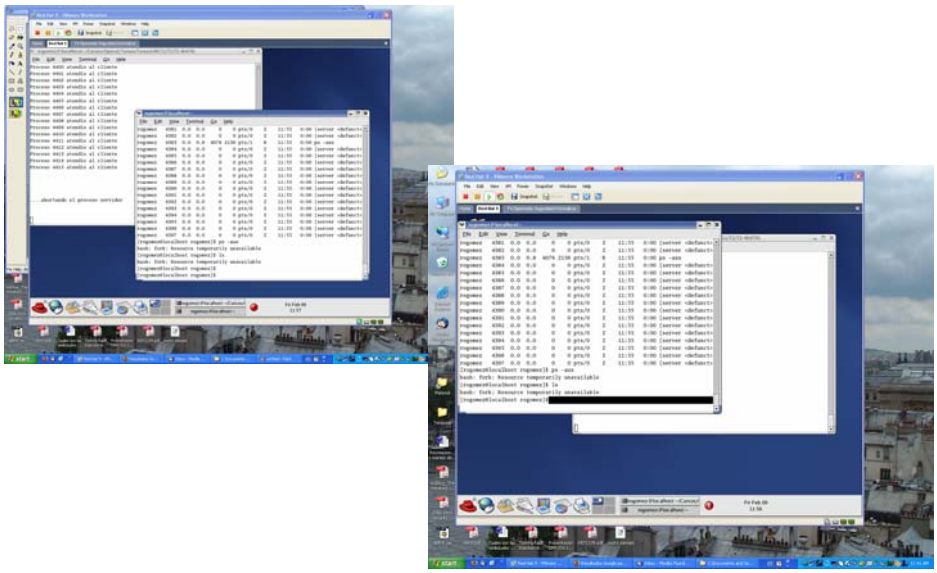


Lámina 6 Roberto Gómez C.

Características Zombies

- Todos tienen el mismo padre
- Aunque los procesos han terminado (exited), poseen una pequeña pieza de información: su status de salida
- Esperan pasarle dicho status a su padre
- Usualmente:
  - padre ejecuta un wait() para sincronizarse con la terminación del hijo
  - el padre muere, y el estatus del hijo es recuperado por el “abuelo” (i.e. shell o init)

Lámina 7 Roberto Gómez C.

Ignorando la señal (SVR4 UNIX)

- Necesario modificar el código del padre
- Ignorar la señal con el parámetro SIG_IGN de la llamada signal()
- Ejemplo:

```
#include <signal.h>
signal(SIGCHLD, SIG_IGN);
```

Lámina 8 Roberto Gómez C.

El código del servidor

```
#include <signal.h>
main( )
{
   signal(SIGCHLD, SIG_IGN);

   sd = socket(AF_UNIX, SOCK_STREAM, 0 );
   bind(sd, (struct sockaddr *)&saddr, addrlen );
   listen(sd, 5);
   while(1)  {
      fd = accept(sd, 0, 0 );
      if ( fork() = = 0 ) {          /*código proceso hijo */
         atencion_servicio(fd);
         exit(0);
      }
      else
         close(fd);                  /* padre no usa conexión */
    }
}
```

Lámina 9 Roberto Gómez C.

Comentarios solución SVR4

- Disposición por default de la señal SIGCHLD es SIG_IGN
  - la llamada no tiene sentido
- Es la forma usual (y documentada) de prevenir la formación de zombies, bajo las circunstancias descritas

Lámina 10 Roberto Gómez C.

Previniendo zombies en BSD

- En sistemas Unix derivados de BSD, la solución anterior no funciona
- Una solución más compleja es necesaria:
  - atrapar la señal SIGCHLD
  - ejecutar una llamada wait()

Lámina 11 Roberto Gómez C.

Primera opción solución

```
#include <signal.h>
void espera()
{
   wait(0);
   signal(SIGCHLD, espera);
}

main( )
{
    signal(SIGCHLD, espera);

    sd = socket(AF_UNIX, SOCK_STREAM, 0 );
    bind(sd, (struct sockaddr *)&saddr, addrlen );
    listen(sd, 5);
    while(1)  {
       fd = accept(sd, 0, 0 );
       if ( fork() = = 0 ) {          /*código proceso hijo */
             :
             :
```

Lámina 12 Roberto Gómez C.

Precauciones a tomar

- Existen llamadas (read(), accept() y select()) que pueden ser interrumpidas si una señal es entregada
- Si esto sucede:
  - llamada regresa un aparente error con el valor EINTR asignado a errno
  - en un servidor el proceso padre estará bloqueado cuando la señal llegue
  - es necesario modificar la llamada para que atrape el regreso de error y restablecer la llamada accept()

Lámina 13 Roberto Gómez C.

Código de la solución

```
#include <signal.h>
      :
      :
void espera()
{
   wait(0);
   signal(SIGCHLD, espera);
}
      :
      :
```

Lámina 14 Roberto Gómez C.

```
main( )
{
    signal(SIGCHLD, espera);
                :
                :
    while (1) {
       ctl_len = sizeof(client);
       DENUEVO:   msgsock= accept(sock, 0, 0)
         if ( msgsock < 0 ) {
            if ( errno = EINTR )    /* accept() interrumpido por la señal */
               goto DENUEVO;   /* por lo que hay intentar de nuevo   */
            else  {
               perror("accept");
               exit(3);
             }
         }
    2
         if ( fork() = = 0 ) {          /*código proceso hijo */
            atencion_servicio(fd);
```

Lámina 15 Roberto Gómez C.

```
    2
                :
                :
         if  ( fork() = = 0 ) {
            close(sock);
            servicio(msgsock)
            exit(0);
         }
         else
             close(msgsock);
    }                                   /* del if ( msgsock < 0 )  */
```

Lámina 16 Roberto Gómez C.

Sockets y threads

llamadas y arquitectura del sistema

Lámina 17 Roberto Gómez C.

Los threads

- Secuencia de pasos de ejecución en un programa.
- Cuenta con su propio contador de programa, así como un stack para llevar un control de variables locales y direcciones de regreso

Lámina 18 Roberto Gómez C.

Secuencia de ejecución de llamada a la función

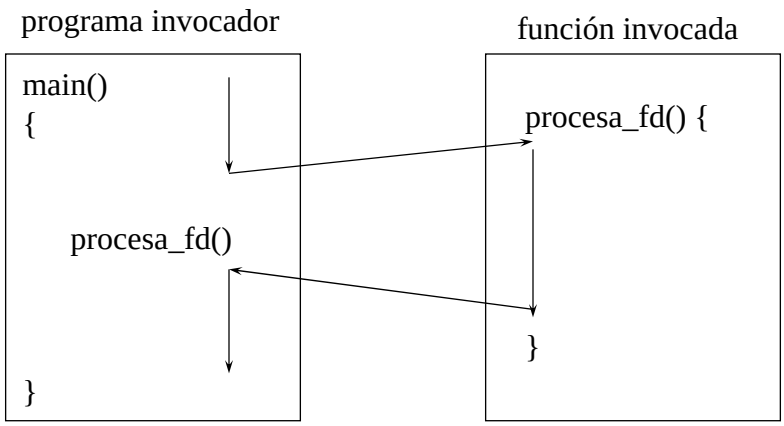


Lámina 19 Roberto Gómez C.

Una nueva secuencia ejecuta la función

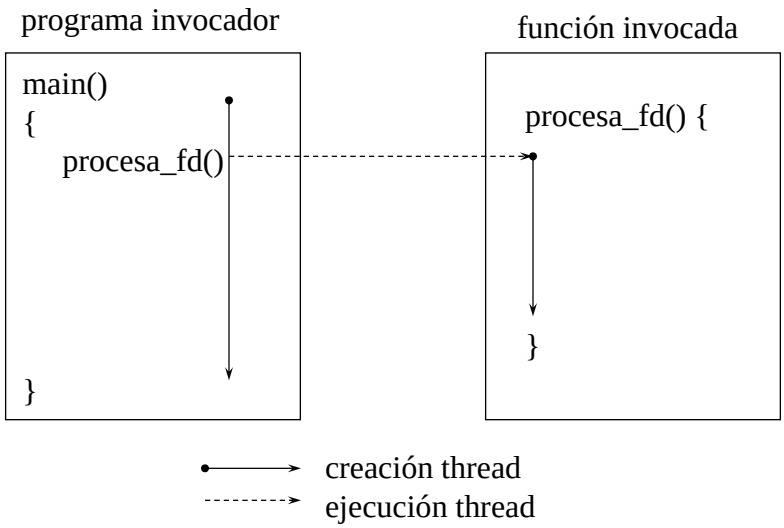


Lámina 20 Roberto Gómez C.

Características de los threads

- Los threads se ejecutan independientemente y simultáneamente
- Comparten instrucciones del proceso y casi todos sus datos
- Si un thread realiza un cambio en una variable en los datos compartidos este será percibido por otros threads en el mismo proceso

Lámina 21 Roberto Gómez C.

Estándares manejo threads

- Programación aplicaciones multithreads es de los 60's
- Desarrollo en sistema Unix: mediados 80´s, después de la salida de Solaris 1
- Solaris proporciona un conjunto de funciones para el manejo de threads, conjuntados en librería *thread.h*
- POSIX (Portable Operating System Interface for uniX) desarrollo su conjunto de funciones agrupandolas en librería *pthread.h*

Lámina 22 Roberto Gómez C.

Llamadas Posix.1c y Sun Solaris 2

Descripción	POSIX	Solaris 2
Manejo Threads	pthread_create pthread_exit pthread_kill pthread_join pthread_self	thr_create thr_exit thr_kill thr_join thr_self
Exclusión Mutua	pthread_mutex_init pthread_mutex_destroy pthread_mutex_lock pthread_mutex_trylock pthread_mutex_unlock	mutex_init mutex_destroy mutex_lock mutex_trylock mutex_unlock

Lámina 23 Roberto Gómez C.

Descripción	POSIX	Solaris 2
Variables de condición	pthread_cond_init pthread_cond_destroy pthread_cond_wait pthread_cond_timedwait pthread_cond_signal pthread_cond_broadcast	cond_init cond_destroy cond_wait cond_timedwait cond_signal cond_broadcast

Lámina 24 Roberto Gómez C.

Creación e inicialización threads

- Rutinas de creación e inicialización
- Se hace referencia a un thread a través de un identificador de tipo *pthread_t*
- Los threads comparten todo el espacio de direcciones del proceso donde son creados

Lámina 25 Roberto Gómez C.

Creando un thread

```
pthread_create(pthread_t  *tid, const pthread_attr_t *attr,
        void *(*rutina)(void *), void *arg);
```

- crea un thread y lo pone en la cola de listos
- regresa 0 si todo salió bien, -1 en caso error
- *pthread_t *tid*: identificador del thread
- *const pthread_attr_t *attr*: atributos thread
- *void (*rutina)(void *)*: función llamada por el thread cuando este comienza ejecución
- *void *arg*: argumento de la función llamada

Lámina 26 Roberto Gómez C.

Ejemplo creación thread

```
#include<stdio.h>
    :
#include<pthread.h>

main()
{
  pthread_t tid;
  int fd;
  if (  (fd = open("toto", O_RDONLY) ) == -1)
          perror("Imposible arbir toto");
  else
        if (pthread_create(&tid, NULL, procesa_fd, (void *)&fd) )
           perror("No se pudo crear el thread ");

                                    :
                                    :
```

Lámina 27 Roberto Gómez C.

Servidores concurrentes con threads

```
sd = socket(AF_UNIX, SOCK_STREAM, 0 );
bind(sd, (struct sockaddr *)&saddr, addrlen );
listen(sd, 5);
while(1)  {
    fd = accept(sd, 0, 0 );
    pthread_create(&tid, NULL, servicio, (void *)fd);
}
```

Lámina 28 Roberto Gómez C.

El servicio del thread

```
void servicio( int *fd)
{
    int sock;
    sock = *fd;
    read(sock, mensaje, sizeof(mensaje));

    xxxxxxx
    xxxxxxxx

   write(sock, respuesta, sizeof(respuesta));
   close(sock);
   pthread_exit(0);   /* NO TERMINAR CON exit(0) */
}
```

Lámina 29 Roberto Gómez C.

Algunas llamadas utiles

- pthread_exit
  - abortar el thread sin matar proceso
- pthread_join
  - esperar a que thread termine
- pthread_self
  - regresa el identificador del thread

Lámina 30 Roberto Gómez C.

Raw Sockets

o lo que es lo mismo… por que usar sockets en C y no en Java

Lámina 31 Roberto Gómez C.

API de sockets

- Introducido en 1981 por Unix BSD 4.1
  - implementado como system calls
  - originalmente para Unix, pero Windows y Linux son parecidos
- Tipos de servicio
  - datagrama (UPD)
  - stream (TCP)
  - raw (IP)

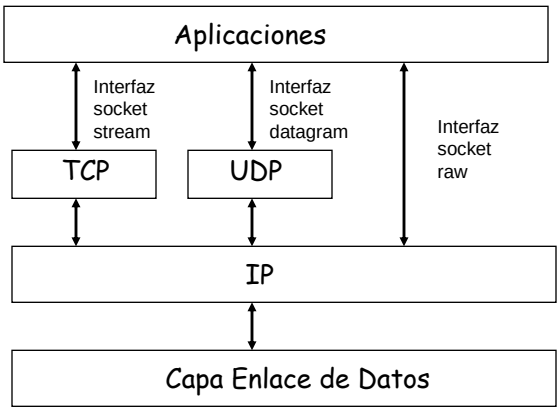
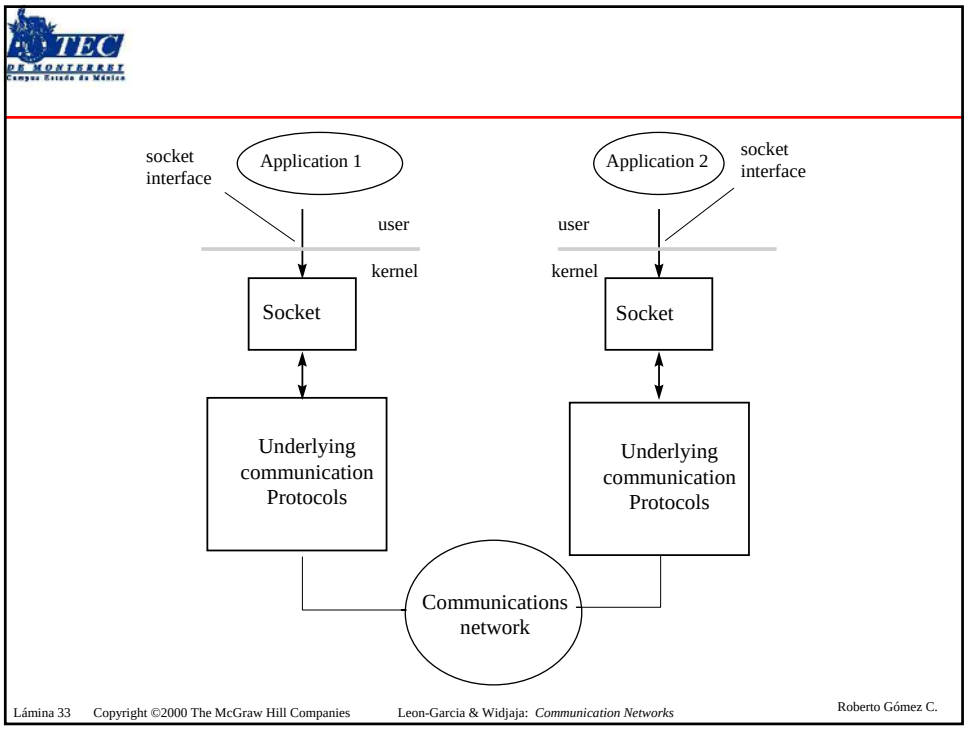


Lámina 32 Roberto Gómez C.



Raw Sockets

- Proporcionan tres características no ofrecidas normalmente por los sockets TCP y UDP
  - Dejan que el programador lea y escriba mensajes ICMP (Internet Control Message Protocol) e IGMP (Internet Group Message Protocol).
  - Aplicaciones pueden leer y escribir datagramas IPv4 sin que el núcleo procese un campo del protocolo IPv4
    - por ejemplo: OSPF con el campo de protocolo con un valor de 89.
  - Un proceso puede construir su propio encabezado IPv4, usando la opción IP_HDRINCL de la llamada setsockopt( )

Lámina 34 Roberto Gómez C.

Usando sockets tipo RAW

- Para usar sockets de tipo raw en Unix es necesario contar con privilegios de root
- Para crear un socket tipo raw es necesario usar la llamada socket() con los siguientes parámetros

  s=socket(AF_INET, SOCK_RAW, [protocolo])
- A partir de esto es posible enviar/recibir sobre este socket.
- Sockets tipo raw pueden usarse para generar/recibir paquetes de un tipo que el kernel no soporta explicitamente.

Lámina 35 Roberto Gómez C.

Valores protocolos

- Valores usados para definir el campo de Protocolo en el encabezado IP

– IP (dummy)	IPPROTO_IP	0
– ICMP	IPPROTO_ICMP	1
– IGMP	IPPROTO_IGMP	2
– Gateway	IPPROTO_GGP	3
– TCP	IPPROTO_TCP	6
– PUP	IPPROTO_PUP	12
– UDP	IPPROTO_UDP	17
– XND IDP	IPPROTO_IDP	22
– Net Disk	IPPROTO_ND	77
– Raw IP	IPPROTO_RAW	255

Lámina 36 Roberto Gómez C.

Opciones llamada setsockopt()

- Usado para modificar opciones sockets
  - `IP_ADD_MEMBERSHIP`
    - Join a multicast group on a specified local interface
  - `IP_DROP_MEMBERSHIP`
    - Leave a multicast group
  - `IP_MULTICAST_IF`
    - Specify the interface for *outgoing* multicast datagrams sent on this socket
  - `IP_MULTICAST_TTL`
    - Set the IPv4 TTL parameter (if not specified, default=1)
  - `IP_MULTICAST_LOOP`
    - Enable or disable local loopback (default is enabled)

Lámina 37 Roberto Gómez C.

La opción IP_HDRINCL de setsockopt( )

- Si la opción `IP_HDRINCL` NO esta activada
  - el núcleo construye el encabezado de IP y pone los datos del proceso después del encabezado
  - él núcleo asigna el campo de protocolo del encabezado IPv4
- Si la opción `IP_HDRINCL` esta activada
  - el proceso construye todo el encabezado IP, excepto el checksum del encabezado IPv4 (siempre calculado por el núcleo) y el campo de identificación IPv4 (si se le asigna un valor de cero, el núcleo lo asigna).
- El núcleo fragmenta paquetes raw que excedan la interfaz de salida MTU

Lámina 38 Roberto Gómez C.

Raw Sockets: socket input

- Paquetes TCP y UDP nunca son entregados a un raw socket
- La mayoría de los paquetes ICMP y todos los paquetes IGMP se pasan a un raw socket
- Si el núcleo no entiende el campo de protocolo de un datagrama IP, se pasa a un raw socket
- Si el datagrama de entrada esta fragmentado, no se pasa nada al raw socket hasta que el reensamblado sea hecho.
- Núcleo examina todos los raw sockets de todos los procesos para decidir a quien se le entregan los datagramas que lleguen.

Lámina 39 Roberto Gómez C.

Internet Control Message Protocol

- ICMP definido en el RFC 792
- Mensajes ICMP
  - Interroga a nodo(s) por información
  - Reporta condiciones de error
- Mensajes ICMP son transportados como datagramas IP
  - ICMP usa o esta debajo de IP
- Mensajes ICMP son procesados por IP, UDP, o TCP
  - IP, TCP, y UDP “usan” o están abajo de ICMP

Lámina 40 Roberto Gómez C.

Ejemplo uso sockets tipo RAW

- Un ejemplo de uso es PING
- Ping trabajo enviando un paquete echo del protocolo ICMP
- El núcleo contiene código para responder a paquetes echo-ping
- No tiene código para generar estos paquetes debido a que no se requiere
- El generador de paquetes pings es un programa que funciona en espacio usuario
  - forma un paquete echo ICMP y lo envía a través de un socket tipo raw, esperando por una respuesta

Lámina 41 Roberto Gómez C.

Ejemplos tipos mensajes ICMP

- Queries
  - TYPE = 8: Echo request
  - TYPE = 0: Echo reply
  - TYPE = 13: Time stamp request
  - TYPE = 14: Time stamp reply
- Errores
  - TYPE = 3: Destination unreachable
    - CODE = 0: Network unreachable
    - CODE = 1: Host unreachable
    - CODE = 2: Protocol unreachable
    - CODE = 3: Port unreachable
  - TYPE = 11: Time exceeded
    - CODE = 0: Time-to-live equals 0 in transit

Lámina 42 Roberto Gómez C.

ICMP dentro del stack TCP/IP

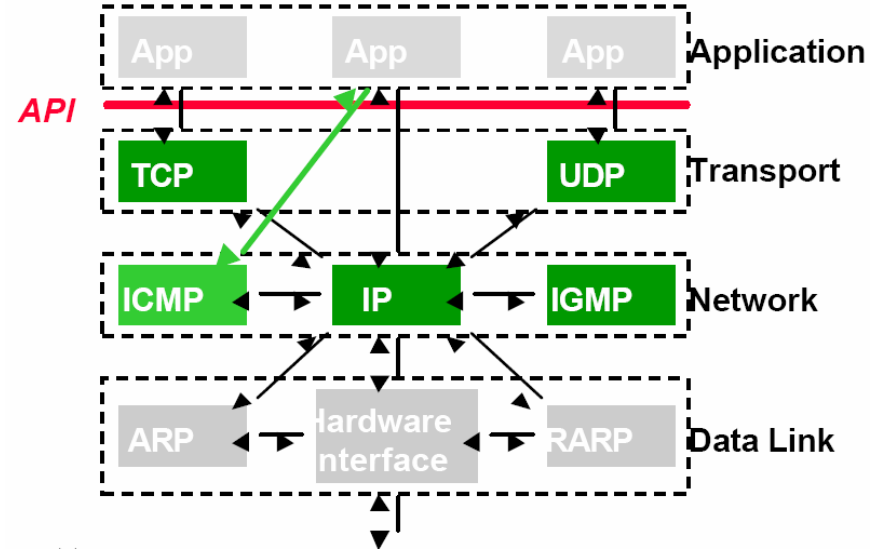


Lámina 43 Roberto Gómez C.

Encabezado protocolo IP

0	3	7	15	18	23	31
número versión	longitud	tipo de servicio	longitud del paquete			
identificación				DF	MF	offset del fragmento
tiempo de vida		transporte	checksum del encabezado			
dirección fuente						
dirección destino						
opciones						relleno

Lámina 44 Roberto Gómez C.

IP HEADER STRUCT

```
struct iphdr {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;
    unsigned int ihl:4;
#else
# error    "Please fix <bits/endian.h>"
#endif
    u_int8_t tos;
    u_int16_t tot_len;
    u_int16_t id;
    u_int16_t frag_off;
    u_int8_t ttl;
    u_int8_t protocol;
    u_int16_t check;
    u_int32_t saddr;
    u_int32_t daddr;
    /*The options start here. */
};
```

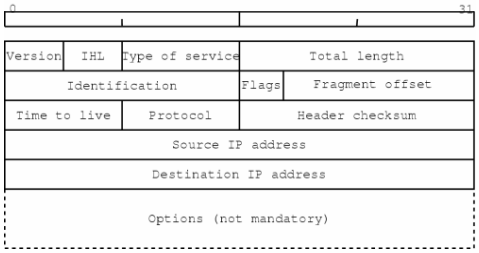


Lámina 45 Roberto Gómez C.

Encabezado TCP

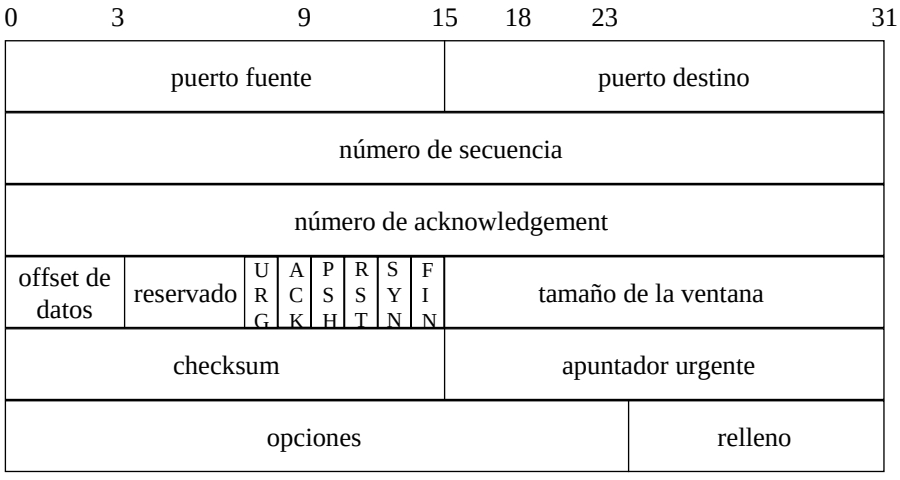


Lámina 46 Roberto Gómez C.

TCP HEADER STRUCTURE

```
struct tcphdr {
        __u16   source;
        __u16   dest;
        __u32   seq;
        __u32   ack_seq;
#if defined(__LITTLE_ENDIAN_BITFIELD)
        __u16   res1:4,
                doff:4,
                fin:1,
                syn:1,
                rst:1,
                psh:1,
                ack:1,
                urg:1,
                res2:2;
#elif defined(__BIG_ENDIAN_BITFIELD)
                __u16   doff:4,
                res1:4,
                res2:2,
                urg:1,
                ack:1,
                psh:1,
                rst:1,
                syn:1,
                fin:1;
#else
#error  "Adjust your <asm/byteorder.h> defines"
#endif
        __u16   window;
        __u16   check;
        __u16   urg_ptr;
};
```

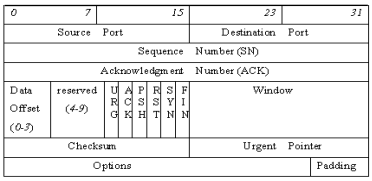
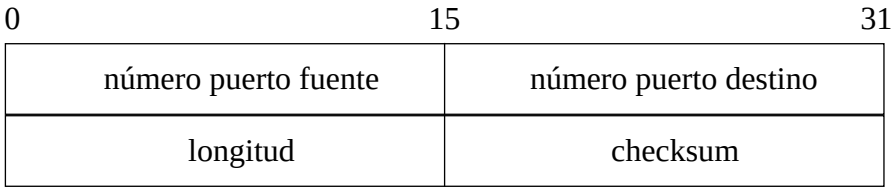


Lámina 47 Roberto Gómez C.

Encabezado y estructura UDP




```
struct udphdr {
        __u16 source;
        __u16 dest;
        __u16 len;
        __u16 check;
};
```

Lámina 48 Roberto Gómez C.

Encabezado y estructura ICMP

```
struct icmphdr {
  __u8          type;
  __u8          code;
  __u16         checksum;
  union {
        struct {
                __u16   id;
                __u16   sequence;
        } echo;
        __u32   gateway;
        struct {
                __u16   __unused;
                __u16   mtu;
        } frag;
  } un;
};
```

Tipo	Codigo	Checksum
Dato ICMP		

Lámina 49 Roberto Gómez C.

Ejemplo mensajes ICMP

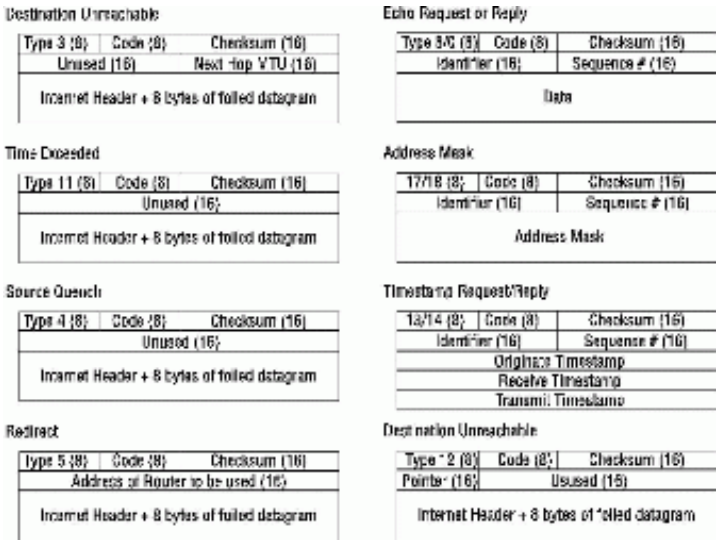


Lámina 50 Roberto Gómez C.

Función cálculo checksum

```
uint16_t in_cksum( uint16_t *addr, int len )
{
  int nleft = len;
  uint32_t sum = 0;
  uint16_t *w = addr;
  uint16_t answer = 0;

  while (nleft > 1)  {
        sum += *w++;
        nleft -= 2;
    }
   if (nleft == 1)    {
        *(u_char *) (&answer) = *(u_char *) w;
        sum += answer;
    }
   sum = (sum >> 16) + (sum & 0xffff);      /* add hi 16 to low 16 */
   sum += (sum >> 16);                      /* add carry */
   answer = ~sum;                           /* truncate to 16 bits */
   return (answer);
}
```

Lámina 51 Roberto Gómez C.

Ejemplo uso raw sockets

```
/*************************************************************************/
/*                                                                       */
/*          Exile 2000 International Coding Team                         */
/*              (http://www.exile2k.org)                                 */
/*            All rights reserved Exile Team                             */
/*            Copyright 2000 (C) Nitr0gen                                */
/*                                                                       */
/*        This function basicly build a ICMP message (PING)              */
/*                including Ip header                                    */
/*                                                                       */
/*        Your almost done! Compile the example and enjoy                */
/*              understanding my rustic code                             */
/*                                                                       */
/*************************************************************************/
```

Lámina 52 Roberto Gómez C.

Encabezados y funciones

```
#include <stdio.h>
#include <stdlib.h>

#include <linux/ip.h>
#include <linux/icmp.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

unsigned short in_cksum(unsigned short *addr, int len);
```

Lámina 53 Roberto Gómez C.

Declaraciones main() e inicializaciones

```
int main()
{
        int sock, optval;
        char *packet, *buffer;
        char *ipF, *ipD;
        struct icmphdr *icmp;
        struct sockaddr_in peer;
        struct iphdr *ip;
        struct in_addr dir1, dir2;
        int r;

            /* reservando espacio para el paquete a enviar */
            packet  = (char *) malloc(sizeof(struct iphdr) + sizeof(struct icmphdr));

            /* reservando espacio para el paquete a recibir */
            buffer  = (char *) malloc(sizeof(struct iphdr) + sizeof(struct icmphdr));
```

Lámina 54 Roberto Gómez C.

Construyendo encabezado IP

```
/* asignado los apuntadores a la direccion donde va a residir el paquete */
ip = (struct iphdr *) packet;
icmp = (struct icmphdr *) (packet + sizeof(struct iphdr));

/* asignando los valores a los campos del encabezado IP */
ip->ihl     = 5;
ip->version = 4;
ip->tos     = 0;
ip->tot_len = sizeof(struct iphdr) + sizeof(struct icmphdr);
ip->id      = htons(getuid());
ip->ttl     = 255;
ip->protocol = IPPROTO_ICMP;
```

Lámina 55 Roberto Gómez C.

Construyendo direcciones encabezado IP

```
/* convirtiendo las direcciones a formato binario */

ipF="10.10.10.10";
ipD="20.20.20.20";
if ( ((inet_aton(ipF, &dir1)) == 0) || ((inet_aton(ipD,&dir2)) == 0) ) {
  perror("inet_aton:");
  exit(1);
}

ip->saddr    = dir1.s_addr;
ip->daddr    = dir2.s_addr;
```

Lámina 56 Roberto Gómez C.

Construyendo socket y activando opción definición encabezado

```
/* creando el socket y asignando la opcion de IP_HDRINCL */
        sock = socket(AF_INET,SOCK_RAW,IPPROTO_ICMP);
        if (sock < 0) {
          perror("Error creacion socket:");
          exit(1);
        }
            r = setsockopt(sock,IPPROTO_IP,IP_HDRINCL,&optval,sizeof(int));
        if (r < 0) {
          perror("setsockopt");
          exit(1);
        }
```

Lámina 57 Roberto Gómez C.

Construyendo encabezado ICMP

```
/* asignando los valores al paquete ICMP*/
icmp->type = ICMP_ECHO;
icmp->code = 0;
icmp->un.echo.id = 0;
icmp->un.echo.sequence = 0;
icmp->checksum = 0;

/* calculo y asignacion del checksum de los paquetes*/
icmp->checksum = in_cksum((unsigned short *)icmp,sizeof(struct icmphdr));
ip->check    = in_cksum((unsigned short *)ip, sizeof(struct iphdr));
```

Lámina 58 Roberto Gómez C.

Enviando el paquete

```
peer.sin_family = AF_INET;
peer.sin_addr.s_addr = dir2.s_addr;

/* envio del paquete ICMP echo*/
 r = sendto(sock,packet,ip->tot_len,0,(struct sockaddr *)&peer,
            sizeof(struct sockaddr));
 if (r < 0) {
     perror("sendto:");
     exit(1);
 }
 printf("Paquete ICMP ECHO enviado \n");
```

Lámina 59 Roberto Gómez C.

Esperando la respuesta

```
/* recepcion de la respuesta del paquete ICMP*/

r = recv(sock,buffer,sizeof(struct iphdr)+sizeof(struct icmphdr),0);
 if (r < 0) {
   perror("recv:");
   exit(1);
 }
 printf("Received the ECHO REPLY\n");
 close(sock);
 return 0;
}
```

Lámina 60 Roberto Gómez C.