Implementação do TCP/IP em BSD4.4

FEUP/MRSC/AMSR
MPR

Bibliografia

» Aula preparada com base nos seguintes documentos

- Gary R. Wright, W. Richard Stevens, “TCP/IP Illustarted, Volume 2 – The Implementation”, Addison- Wesley, 1995
- Samuel J. Leffler, William N. Joy, Robert S. Fabry and Michael J. Karels, “Networking Implementation Notes 4.4BSD Edition”, Technical Report, 1993.

» Nota

- Os conceitos arquitectónicos apresentados nesta aula são genéricos e não pretendem descrever com rigor nenhuma variante do BSD4.4.
- BSD – Berkeley Software Distribution

Subsistema de Rede do BSD4.4

» Software implementado
- Parte da camada de sessão (nível 5 do modelo OSI)
- Camada de transporte (4)
- Camadas de rede (1, 2, 3)

» Projectado para suporte de múltiplas famílias de protocolos
- Representação flexível dos dados ➔Recurso intensivo a *casting* (ex. sockaddr)
- Variáveis partilhadas apenas por entidades
  - da mesma família de protocolos
  - que comunicam por *rendez-vous*

» Pilhas suportadas
- TCP/IP
- XNS (Xerox Network Systems)
- OSI
- Protocolos do domínio UNIX

Representação Interna de Endereços

» Endereços → associados a 1 família de protocolos
- *sa_family* → família representada
- *sa_data* → endereço

```
struct sockaddr {
        short sa_family;    /* data format identifier */
        char  sa_data[14];  /* address */
};
```

sa_family	Família de Protocolos
AF_INET	Internet
AF_ISO, AF_OSI	OSI
AF_UNIX	Unix
AF_ROUTE	Routing table
AF_LINK	Data link

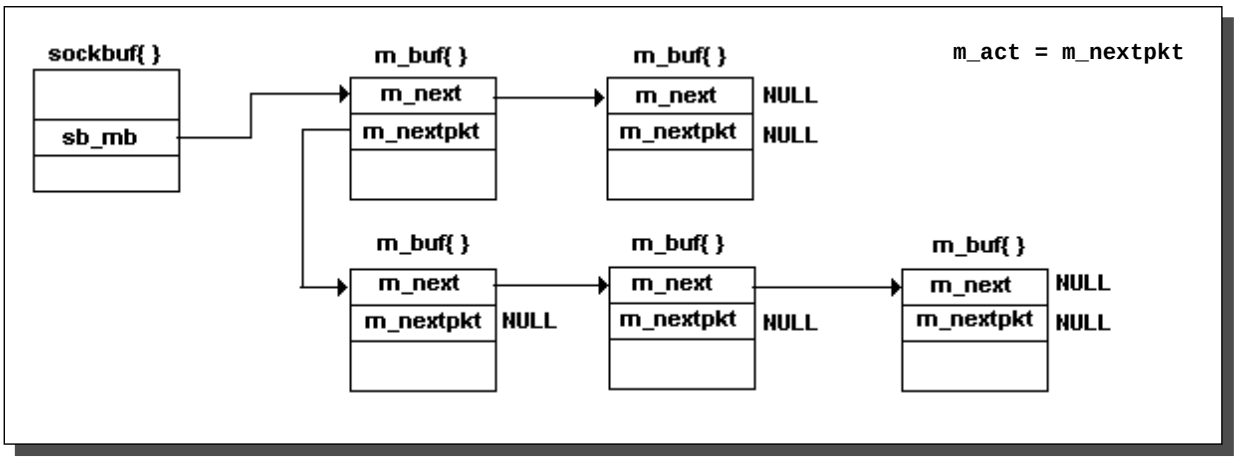
Memory Buffers

```
struct mbuf {
     struct mbuf *m_next;  /* next buffer in chain */
     u_long m_off;         /* offset of data */
     short  m_len;         /* amount of data in this mbuf */
     short  m_type;        /* mbuf type (accounting) */
     u_char m_dat[MLEN];   /* data storage */
     struct mbuf *m_act;   /* link in higher-level mbuf list */
};
```

» Usado para armazenamento de informação temporária
  – dados
  – pacotes, cabeçalhos de pacotes
  – estatísticas de utilização ➔ observadas com comando *netstat*

» comprimento constante (128 bytes)
  – `m_next` → lista ligada de `mbufs` (cadeia) → ex.: um pacote de dados
  – `m_act` → lista de cadeias → ex.: fila de pacotes
  – `m_dat` → armazenamento de informação → ex.: parte de um pacote
  – `m_len` → comprimento dos dados usado
  – `m_off` → distância da base do *mbuf* ao início dos dados

Listas de Cadeias de **`mbuf`**

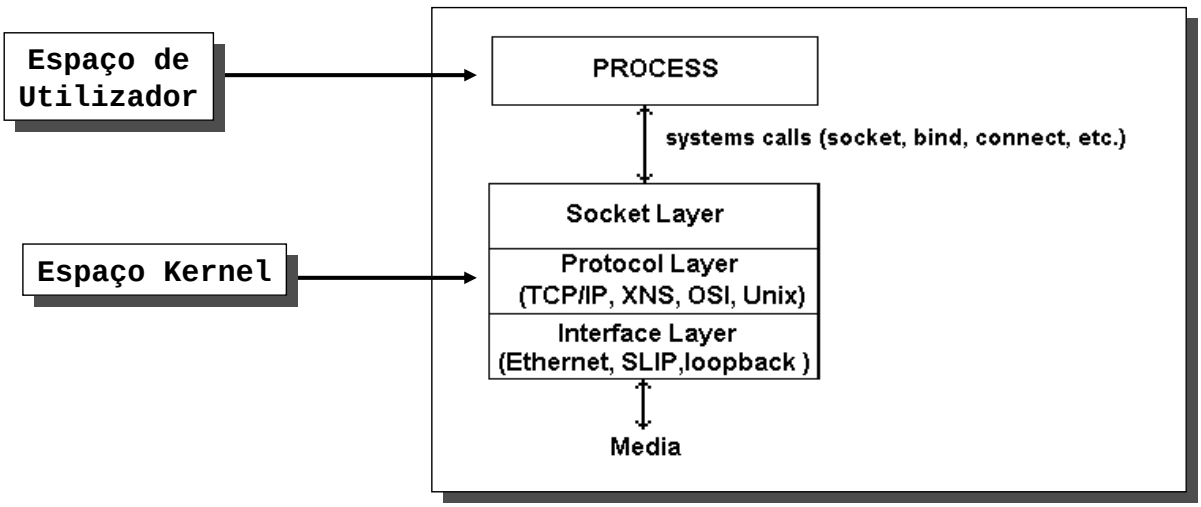


Manipulação de **mbuf**

m = m_get(wait,type)	Allocates an mbuf, placing its address in m. The argument wait is either M_WAIT or M_DONTWAIT according to whether allocation should block or fail if no mbuf is available. The type is one of the predefined mbuf types for use in accounting ofmbuf allocation.
n = m_fre(m)	Frees a single mbuf, m, placing a pointer to its successor in the chain it heads, if any, in n.
m_freem(m)	Frees an mbuf chain headed by m.
m = m_copy(m0, off, len)	Creates a copy of all, or part, of a list of the mbufs in m0. Len bytes of data, starting off bytes from the front of the chain, are copied.
m_cat(m, n)	The mbuf chain, n, is appended to the end of m. Where possible, compaction is performed
m_adj(m, diff)	The mbuf chain, m, is adjusted in size by diff bytes. If diff is non-negative, diff bytes are shaved off the front of the mbuf chain. If diff is negative, the alteration is performed from back to front. No space is reclaimed in this operation; alterations are accomplished by changing the m_len and m_off fields of mbufs
m = m_pullup(m0, size)	After a successful call to m_pullup, the mbuf at the head of the returned list, m, is guaranteed to have at least size bytes of data in contiguous memory within the data area of the mbuf (allowing access via a pointer and allowing the mbuf to be located from a pointer to the data area). If the original data was less than size bytes long,len was greater than the size of an mbuf data area (112 bytes), or required resources were unavailable, m is 0 and the original mbuf chain is deallocated.

Estrutura do Sistema de Rede

» 3 camadas
- Camada de Sockets
- Camada de Protocolos → organizados em módulos
- Camada de Interfaces de rede



Socket

```
struct socket {
     short   so_type;          /* generic type */
     short   so_options;       /* from socket call */
     short   so_linger;        /* time to linger while closing */
     short   so_state;         /* internal state flags */
     caddr_t so_pcb;           /* protocol control block */
     struct  protosw *so_proto; /* protocol handle */
     struct  socket *so_head;  /* back pointer to accept socket */
     struct  socket *so_q0;    /* queue of partial connections */
     short   so_q0len;         /* partials on so_q0 */
     struct  socket *so_q;     /* queue of incoming connections */
     short   so_qlen;          /* number of connections on so_q */
     short   so_qlimit;        /* max number queued connections */
     struct  sockbuf so_rcv;   /* receive queue */
     struct  sockbuf so_snd;   /* send queue */
     short   so_timeo;         /* connection timeout */
     u_short so_error;         /* error affecting connection */
     u_short so_oobmark;       /* chars to oob mark */
     short   so_pgrp;          /* pgrp for signals */
};
```


Estados do Socket

```
#define SS_NOFDREF              0x001 /* no file table ref any more */
#define SS_ISCONNECTED          0x002 /* socket connected to a peer */
#define SS_ISCONNECTING         0x004 /* in process of connecting to peer */
#define SS_ISDISCONNECTING      0x008 /* in process of disconnecting */
#define SS_CANTSENDMORE         0x010 /* can't send more data to peer */
#define SS_CANTRCVMORE          0x020 /* can't receive more data from peer */
#define SS_RCVATMARK            0x040 /* at mark on input */
#define SS_PRIV                 0x080 /* privileged */
#define SS_NBIO                 0x100 /* non-blocking ops */
#define SS_ASYNC                0x200 /* async i/o notify */
```

Estado do **Socket** → alterável por utilizadores e protocolos

Filas do Socket

- Dados armazenados em 2 filas ➔ **so_rcv**, **so_snd** ➔ cadeias de mbufs
- Controlo de fluxo por parte dos processos
  - Utilização de marcas de água ➔ **sb_hiwat**, **sb_lowat**
  - Número de caracteres no buffer ➔ **sb_cc**
- Processos cooperam no controlo de fluxo. Processo bloqueia quando
  - escreve dados no socket **&& sb_cc >= sb_hiwat**
  - lê dados do socket **&& sb_cc <= sb_lowat**

```
struct sockbuf {
     u_short sb_cc;         /* actual chars in buffer */
     u_short sb_hiwat;      /* max actual char count */
     u_short sb_mbcnt;      /* chars of mbufs used */
     u_short sb_mbmax;      /* max chars of mbufs to use */
     u_short sb_lowat;      /* low water mark */
     short   sb_timeo;      /* timeout */
     struct  mbuf *sb_mb;   /* the mbuf chain */
     struct  proc *sb_sel;  /* process selecting read/write */
     short   sb_flags;      /* flags, see below */
};
```


Filas do Socket

» 2 formas de armazenar dados nos buffers

- *Stream-oriented*
  - Sem endereços ou cabeçalhos
  - Dados armazenados em mbufs ligados por **m_next**
- *Record-oriented*
  - Dados armazenados como listas de pacotes
  - Utilização de **m_next** e **m_act**
- Exclusividade no acesso (processo, kernel) garantida por **sb_flags**

```
#define  SB_LOCK 0x01  /* lock on data queue (so_rcv only) */
#define  SB_WANT 0x02  /* someone is waiting to lock */
#define  SB_WAIT 0x04  /* someone is waiting for data/space */
#define  SB_SEL  0x08  /* buffer is selected */
#define  SB_COLL 0x10  /* collision selecting */
```

Protocolos de Comunicação

» Socket criado num ***domínio*** de comunicações ➔ família de endereços

```
struct domain {
     int   dom_family;           /* PF_xxx */
     char  *dom_name;
     int   (*dom_init)();        /* initialize domain data structures */
     int   (*dom_externalize)(); /* externalize access rights */
     int   (*dom_dispose)();     /* dispose of internalized rights */
     struct protosw *dom_protosw, *dom_protoswNPROTOSW;
     struct domain *dom_next;
};
```



```
struct protosw {
     short  pr_type;              /* socket type used for */
     struct domain *pr_domain;    /* domain protocol a member of */
     short  pr_protocol;          /* protocol number */
     short   pr_flags;            /* socket visible attributes */
/* protocol-protocol hooks */
     int    (*pr_input)();        /* input to protocol (from below) */
     int    (*pr_output)();       /* output to protocol (from above) */
     int    (*pr_ctlinput)();     /* control input (from below) */
     int    (*pr_ctloutput)();    /* control output (from above) */
/* user-protocol hook */
     int    (*pr_usrreq)();       /* user request */
/* utility hooks */
     int    (*pr_init)();         /* initialization routine */
     int    (*pr_fasttimo)();     /* fast timeout (200ms) */
     int    (*pr_slowtimo)();     /* slow timeout (500ms) */
     int    (*pr_drain)();        /* flush any excess space possible */
};
```

```
#define PR_ATOMIC       0x01  /* exchange atomic messages only */
#define PR_ADDR         0x02  /* addresses given with messages */
#define PR_CONNREQUIRED 0x04  /* connection required by protocol */
#define PR_WANTRCVD     0x08  /* want PRU_RCVD calls */
#define PR_RIGHTS       0x10  /* passes capabilities */
```

Protocolos

- Protocolo iniciado por **`pr_init`**
- Chamado com período de
  - 200 ms, por **`pr_fasttimo`**
  - 500 ms, por **`pr_slowtimo`**
- Protocolos trocam
  - » dados como cadeias de **`mbufs`**, por
    - **`pr_input`** ➔ dados para cima (em direcção ao utilizador)
    - **`pr_output`** ➔ dados para baixo (em direcção à rede)
  - » informação de controlo por
    - **`pr_ctlinput`**
    - **`pr_ctloutput`**
- Interface com os sockets ➔ **`pr_usrreq`**

Interface de Rede

- » Define *saída* para os pacotes
- » Gere o controlador de rede (hardware)
- » Encapsula, desencapsula cabeçalho do nível 2
- » Tem endereços associados ➔ nível 3, nível 2

- » Constítuida por
  - Uma fila para envio de pacotes
  - Funções
    - `if_init` ➔ inicialização da interface
    - `if_output` ➔ envio de pacotes
    - `if_watchdog` ➔ de timer
      - Periodicamente → decrementa `if_timer`
      - Quando if_timer = = 0 → chama `if_watchdog`

Interface de Rede - **ifnet**

```
struct ifnet {
     char   *if_name;            /* name, e.g. ``en'' or ``lo'' */
     short  if_unit;             /* sub-unit for lower level driver */
     short  if_mtu;              /* maximum transmission unit */
     short  if_flags;            /* up/down, broadcast, etc. */
     short  if_timer;            /* time 'til if_watchdog called */
     struct ifaddr *if_addrlist; /* list of addresses of interface */
     struct ifqueue if_snd;      /* output queue */
     int    (*if_init)();        /* init routine */
     int    (*if_output)();      /* output routine */
     int    (*if_ioctl)();       /* ioctl routine */
     int    (*if_reset)();       /* bus reset routine */
     int    (*if_watchdog)();    /* timer routine */
     int    if_ipackets;         /* packets received on interface */
     int    if_ierrors;          /* input errors on interface */
     int    if_opackets;         /* packets sent on interface */
     int    if_oerrors;          /* output errors on interface */
     int    if_collisions;       /* collisions on csma interfaces */
     struct ifnet *if_next;
};
```


Endereços de Uma Interface de Rede

```
struct ifaddr {
       struct sockaddr ifa_addr; /* address of interface */
       union  {
                struct sockaddr  ifu_broadaddr;
                struct sockaddr  ifu_dstaddr;
       } ifa_ifu;
       struct ifnet *ifa_ifp;    /* back-pointer to interface */
       struct ifaddr *ifa_next;  /* next address for interface */
};
#define ifa_broadaddr  ifa_ifu.ifu_broadaddr  /* broadcast address */
#define ifa_dstaddr ifa_ifu.ifu_dstaddr       /* other end of p-to-p link */
```

Estados de uma Interface de Rede

```
#define  IFF_UP           0x1    /* interface is up */
#define  IFF_BROADCAST    0x2    /* broadcast is possible */
#define  IFF_DEBUG        0x4    /* turn on debugging */
#define  IFF_LOOPBACK     0x8    /* is a loopback net */
#define  IFF_POINTOPOINT  0x10   /* interface is point-to-point link */
#define  IFF_NOTRAILERS   0x20   /* avoid use of trailers */
#define  IFF_RUNNING      0x40   /* resources allocated */
#define  IFF_NOARP        0x80   /* no address resolution protocol */
```


Interface – Processos ← → Sockets

- Estruturas de dados de endereços
  - » BSD

```
struct sockaddr {
       u_short sa_family;    /*Address family - ex: AF_INET*/
       char    sa_data[14];  /*Protocol address*/
};
```

  - » Internet

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

Funções Usadas por Clientes e Servidores

➔ `int socket(int family, int type, int protocol)`

family: AF_INET, AF_UNIX
type: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW
protocol: protocolo a usar (com o valor 0 é determinado pelo sistema)

» Retorno
  – descritor de *socket*
  – -1, em caso de erro

➔ `int bind(int sockfd, struct sockaddr* myaddr, int addrlen)`

sockfd: descritor do *socket*
myaddr: endereço local (IP + porto)
addrlen: comprimento da estrutura myaddr

» Retorno
  – 0 em caso de sucesso
  – -1 em caso de erro

» Esta primitiva associa o *socket* ao endereço local myaddr

Funções Usadas por Clientes e Servidores

➔ `int connect(int sockfd, struct sockaddr* serveraddr, int addrlen)`

serveraddr: endereço do servidor remoto (IP + porto)

» Retorno
  – 0 em caso de sucesso
  – -1 em caso de erro

» TCP: estabelecimento de ligação com servidor remoto

» UDP: armazenamento do endereço *serveraddr*

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

backlog: número de pedidos de ligação em fila de espera

» Retorno
  – 0 em caso de sucesso
  –-1 em caso de erro

» Primitiva especifica o número máximo de ligações em fila de espera

Funções Usadas por Clientes e Servidores

```
➔ int accept(int sockfd, struct sockaddr* peeraddr, int* addrlen)
```

peeraddr: estrutura usada para armazenar o endereço do cliente (IP + porto)

addrlen: apontador para o comprimento da estrutura peeraddr

» Retorno
  – descritor do *socket* aceite, endereço do cliente e respectivo comprimento
  – -1 em caso de erro

» Primitiva atende pedido de ligação e cria outro *socket* com as mesmas propriedades que o sockfd

```
➔ int send(int sockfd, const void* buf, int len, unsigned int flags)
➔ int recv(int sockfd, void* buf, int len, unsigned int flags)
```

buf: apontador para a posição de memória que contém/vai conter os dados

flags: MSG_OOB, MSG_PEEK, MSG_DONTROUTE

»Retorno
  – número de octetos escritos/lidos
  – 0 em caso de a ligação ter sido fechada
  –-1 em caso de erro

»Estas primitivas permitem o envio e a recepção de dados da rede

Funções Usadas por Clientes e Servidores

```
➔ int sendto(int sockfd, const void* buf, int len,
          unsigned int flags,
          struct sockaddr* to, int tolen)

➔ int recvfrom(int sockfd, void* buf, int len,
            unsigned int flags,
            struct sockaddr* from, int* fromlen)
```

» to: endereço do destinatário do pacote

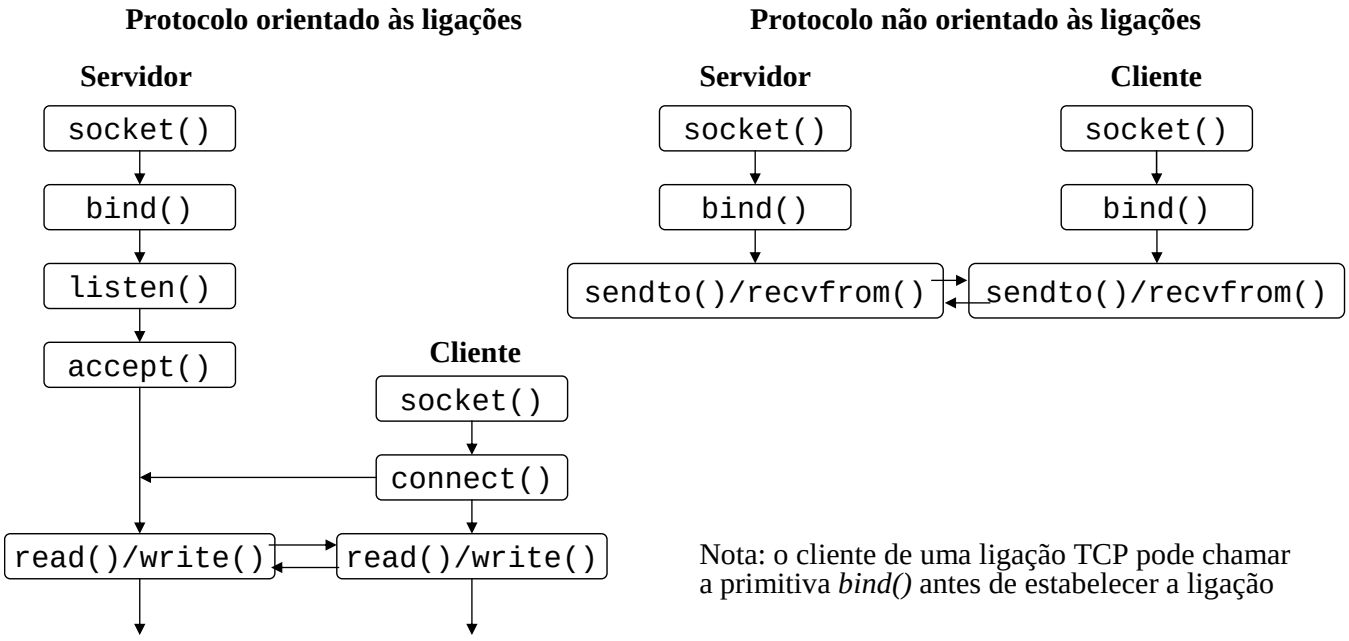
» from: endereço do emissor presente no pacote recebido

» estas primitivas são semelhantes ao send()/recv() mas permitem adicionalmente o envio de mensagens em cenários *connectionless* (UDP), sem haver portanto estabelecimento de ligação

```
➔ int close(int sockfd)
```

» esta primitiva é usada para fechar o socket

Funções Usadas por Clientes e Servidores



Nota: o cliente de uma ligação TCP pode chamar a primitiva *bind()* antes de estabelecer a ligação

Interfaces – Socket ← → Protocolos

» Socket chama função **`pr_usrreq`**

```
int      error;
struct   socket *so;
int      req;
struct   mbuf *m, *addr, *rights;
error =  (*protosw[].pr_usrreq)(so, req, m, addr, rights);
```

Tipos de Pedido

```
#define  PRU_ATTACH      0    /* attach protocol */
#define  PRU_DETACH      1    /* detach protocol */
#define  PRU_BIND        2    /* bind socket to address */
#define  PRU_LISTEN      3    /* listen for connection */
#define  PRU_CONNECT     4    /* establish connection to peer */
#define  PRU_ACCEPT      5    /* accept connection from peer */
#define  PRU_DISCONNECT  6    /* disconnect from peer */
#define  PRU_SHUTDOWN    7    /* won't send any more data */
#define  PRU_RCVD        8    /* have taken data; more room now */
#define  PRU_SEND        9    /* send this data */
#define  PRU_ABORT       10   /* abort (fast DISCONNECT, DETATCH) */
#define  PRU_CONTROL     11   /* control operations on protocol */
#define  PRU_SENSE       12   /* return status into m */
#define  PRU_RCVOOB      13   /* retrieve out of band data */
#define  PRU_SENDOOB     14   /* send out of band data */
#define  PRU_SOCKADDR    15   /* fetch socket's address */
#define  PRU_PEERADDR    16   /* fetch peer's address */
#define  PRU_CONNECT2    17   /* connect two sockets */
/* begin for protocols internal use */
#define  PRU_FASTTIMO    18   /* 200ms timeout */
#define  PRU_SLOWTIMO    19   /* 500ms timeout */
#define  PRU_PROT ORCV   20   /* receive from below */
#define  PRU_PROT OSEND  21   /* send to below */
```


Tipos de Pedido

PRU_ATTACH	When a protocol is bound to a socket (with socket). It is the responsibility of the protocol module to allocate any resources necessary.
PRU_DETACH	Used at the time a socket is deleted. The protocol module may deallocate any resources assigned to the socket
PRU_BIND	Indicates that an address should be bound to an existing socket.
PRU_LISTEN	Indicate the user wishes to listen for incoming connection requests on the asso-ciated socket. The protocol module should perform any state changes needed to carry out thisrequest (if possible).
PRU_CONNECT	Indicate the user wants to a establish an association. The addr parameters upplied describes the peer to be connected to. The effect of a connect request may vary dependingon the protocol. Virtual circuit protocols, such as TC, use this request to initiate estalishment of a TCP connection. Datagram protocols, such as UDP, simply record the peer's address in a private data structure and use it to tag all outgoing packets.
PRU_ACCEPT	Used to indicat the user has accepted the first connection on the queue of pending connections.The protocol module should fill in the supplied address buffer with the address of the connected party
PRU_DISCONNECT	Eliminates an association created with a PRU_CONNECT request. The protocol may deallocate any data structures.
PRU_SEND	Each user request to send data is translated into one or more PRU_SEND requests (a protocol may indicate that a single user send request must be translated into a single PRU_SEND request by specifying the PR_ATOMIC flag in its protocol description). The data to be sent is presented to the protocol as a list of mbufs and an address is, optionally, supplied in the addr parameter. The protocol is responsible for preserving the data in the socket's send queue if it is not able to send it immediately, or if it may need it at some later time
PRU_ABORT	Indicate an abnormal termination of service. The protocol should delete any existing association(s).

Interface – Protocolo ← → Protocolo

» Interface entre módulos de protocolos
- **`pr_input`**
- **`pr_output`**
- **`pr_ctlinput`**
- **`pr_ctloutput`**

`pr_output` *(para baixo)*

» Chamada do UDP para envio de dados

```
int     error;
struct  inpcb * inp;
struct  mbuf * m;
Error = udp_output( inp, m )
```

- `inp` (internet protocol contro block) → informação sobre a ligação (portas e endereços de rede)
- `m` → `Cadeia de mbuf apontada por m` dados a enviar

» Chamada ao IP para envio de dados

```
int     error;
struct  mbuf *m, *opt;
struct  route * ro;
int     flags;
error = ip_output(m, opt, ro, flags);
```

- `m` → dados a enviar
- `opt` → lista dos parâmetros opcionais (a incluir no cabeçalho)
- `ro` → apontador para tabelas de routing
- `flags` → autorização para broadcast e encaminhamento

`pr_input` *(para cima)*

» UDP e TCP

```
struct   mbuf *m;
struct   ifnet *ifp;
(void)   (*protosw[].pr_input)(m, ifp);
```

- `m` → dados como lista de mbufs
- `ifp` → interface de proveniência do pacote

» **`IP, pr_input`** → chamada por uma interrupção de software (sinal)

- Sem parâmetros
- Comunicação com a interface de rede através de uma fila → **`ipintrq`**
  (Semelhante às filas das interfaces de rede)
- Rotina chamada (sinal enviado) quando interface de rede coloca pacote em **`ipintrq`**

`pr_ctlinput`

» Envio de informação de controlo a um módulo de protocolos

- `req` → o pedido feito (maioria são ICMP)
- `addr` → endereço

```
int      req;
struct   sockaddr *addr;
(void) (*protosw[].pr_ctlinput)(req, addr);
```

```
#define  PRC_IFDOWN           0     /* interface transition */
#define  PRC_ROUTEDEAD        1     /* select new route if possible */
#define  PRC_QUENCH           4     /* some said to slow down */
#define  PRC_MSGSIZE          5     /* message size forced drop */
#define  PRC_HOSTDEAD         6     /* normally from IMP */
#define  PRC_HOSTUNREACH      7     /* ditto */
#define  PRC_UNREACH_NET      8     /* no route to network */
#define  PRC_UNREACH_HOST     9     /* no route to host */
#define  PRC_UNREACH_PROTOCOL 10    /* dst says bad protocol */
#define  PRC_UNREACH_PORT     11    /* bad port # */
#define  PRC_UNREACH_NEEDFRAG 12    /* IP_DF caused drop */
#define  PRC_UNREACH_SRCFAIL  13    /* source route failed */
#define  PRC_REDIRECT_NET     14    /* net routing redirect */
#define  PRC_REDIRECT_HOST    15    /* host routing redirect */
#define  PRC_REDIRECT_TOSNET  14    /* redirect for type of service & net */
#define  PRC_REDIRECT_TOSHOST 15    /* redirect for tos & host */
#define  PRC_TIMXCEED_INTRANS 18    /* packet lifetime expired in transit */
#define  PRC_TIMXCEED_REASS   19    /* lifetime expired on reass q */
#define  PRC_PARAMPROB        20    /* header incorrect */
```

Interface – Protocolo ← → Interface de Rede

» Interface entre módulo de protocolo mais baixo e interface de rede

» Envio e recepção de pacotes

» Decisões de encaminhamento tomadas pelos protocolos

Envio de Pacote

» Protocolo chama função de output da interface

```
int      error;
struct   ifnet *ifp;
struct   mbuf *m;
struct   sockaddr *dst;
error =  (*ifp->if_output)(ifp, m, dst)
```

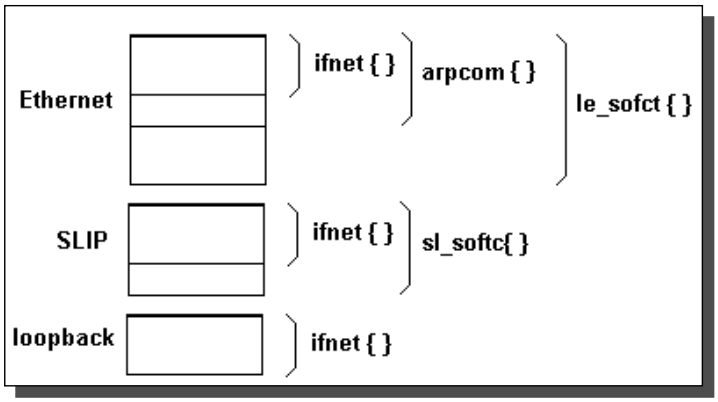
» Pacote **`m`** é enviado para endereço **`dst`**:
- Pacote colocado na fila `send` da interface
- Interface informada
- Erros retornados não são erros de transmissão

Recepção de Pacote

» Pacotes que chegam da rede são colocados pela interface na fila dos protocolos
  – **ipintrq, no `caso do TCP/IP`**
  – Enviado sinal ao módulo do protocolo

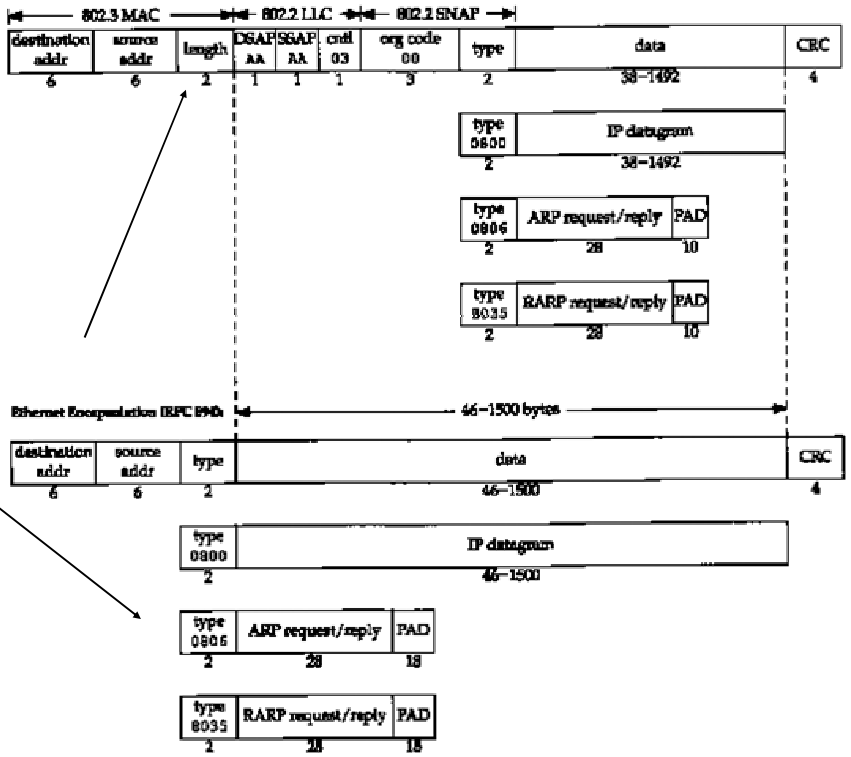
» Usadas macros para introduzir e retirar pacotes das filas. Ex:
  – **`IF_ENQUEUE(ifq, m)`**
    – Coloca pacote `m` no fim da fila **ifq**.
  – **`IF_DEQUEUE(ifq, m)`**
    – Coloca em `m` o apontador para a cadeia de `mbufs` que contém o pacote e retira-o da fila

Especialização das Interfaces de Rede



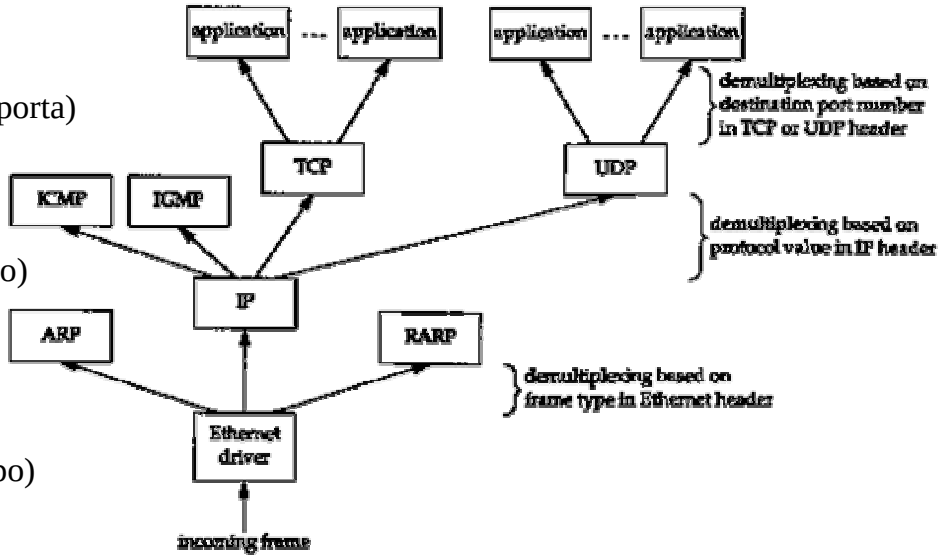
Encapsulamento Ethernet

- Cartas Ethernet
  - » Devem receber
    - – encapsulamento IEEE 802
    - – encapsulamento Ethernet
  - » Se conseguem enviar os 2
    - – encapsulamento Ethernet
- Valores validos IEEE 802 *l*
  - » Diferentes de *type* válidos
    - – Ex. 0800 = 2048

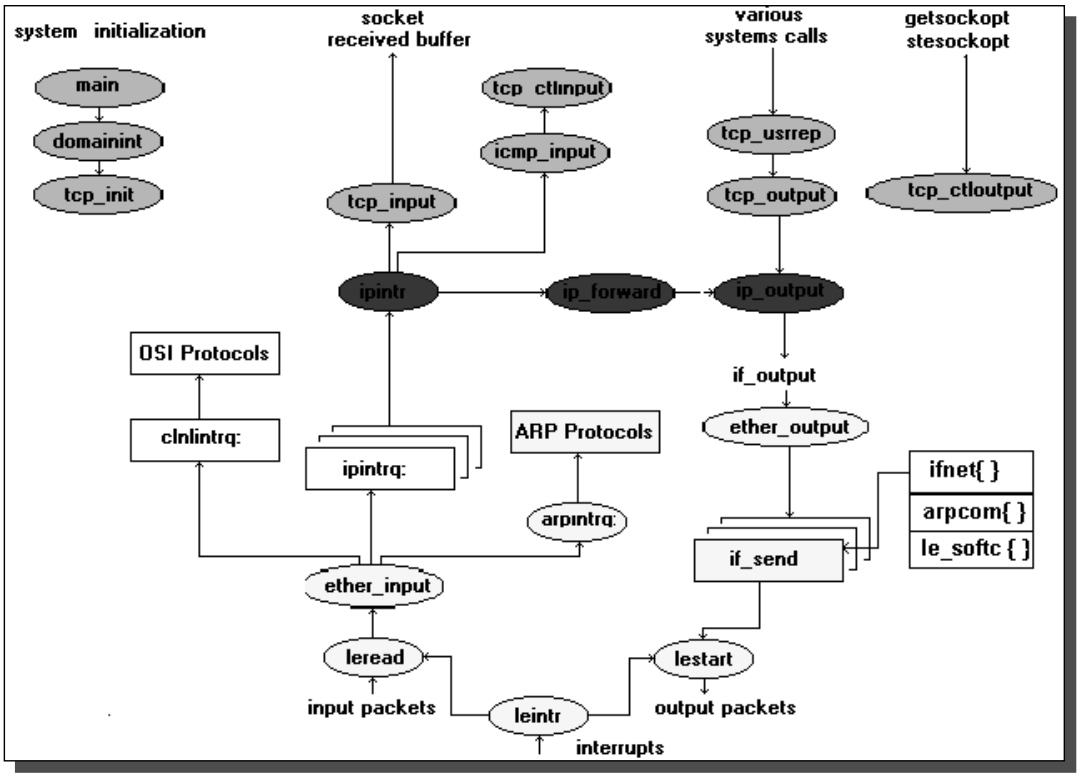


Desmultiplexagem

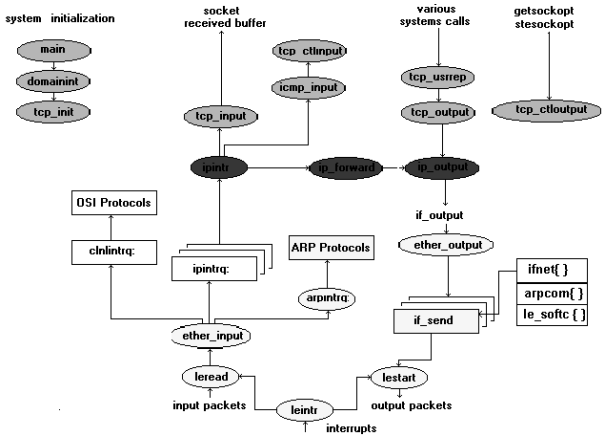
- » Cabeçalho TCP/UDP (porta)
  - – FTP → 21
  - – Telnet → 23
  - – ...
- » Cabeçalho IP (protocolo)
  - – ICMP → 1
  - – IGMP → 2
  - – TCP → 6
  - – UDP → 17
- » Cabeçalho Ethernet (tipo)
  - – IP → 0x0800
  - – ARP → 0x0806
  - – RARP → 0x8053



TCP/IP

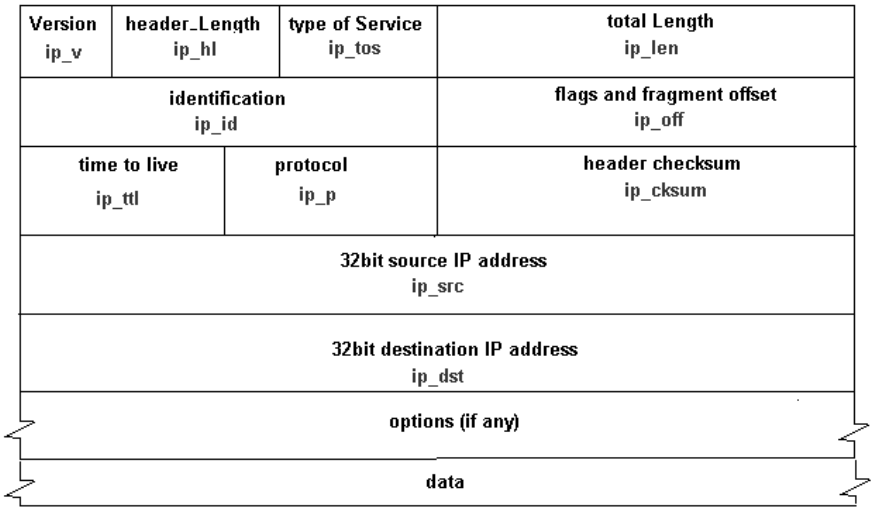


Funções da Interface de Rede Ethernet



- **`leintr`**
  - Rotina associada à interrupção de hardware do controlador Ethernet
  - Chamada quando há recepção (chama `leread`) ou envio (chama `lestart`) de trama
- **`leread`**
  - Transfere trama do controlador para uma cadeia de `mbuf`
- **`ether_input`**
  - Examina cabeçalho da trama ethernet, determina tipo de trama
  - Desmutiplexa a trama, enviando-a para fila correspondente (IP, ARP, ...)
- **`ether_output`**
  - Aceita datagrama, constroi trama ethernet, mete-a na fila do controlador
- **`lestart`**
  - Retira trama da fila e transmite-a. Esvazia a fila

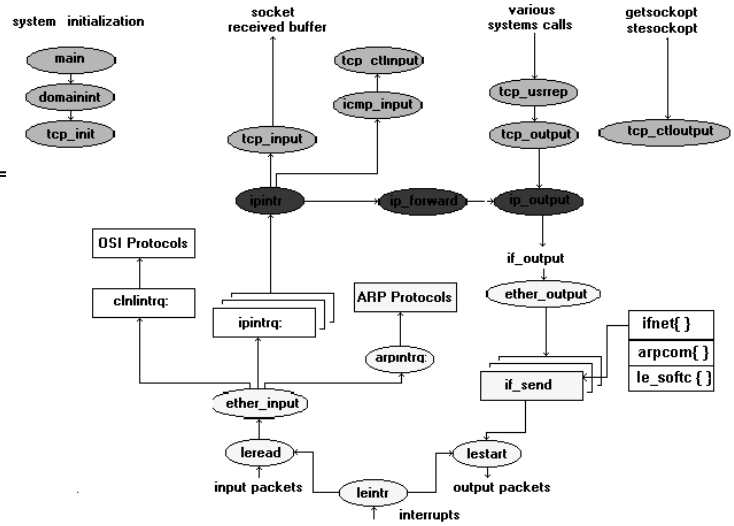
Datagrama IP



Cabeçalho de Datagrama IP (sem opções, em C)

```
struct  ip {
#if BYTE_ORDER == LITTLE_ENDIAN
   u_char  ip_hl:4,         /* header length */
           ip_v:4;          /* version */
#endif
#if BYTE_ORDER == BIG_ENDIAN
   u_char  ip_v:4,          /* version */
           ip_hl:4;         /* header length */
#endif
   u_char ip_tos;           /* type of service */
   short   ip_len;          /* total length */
   u_short ip_id;           /* identification */
   short   ip_off;          /* fragment offset field */
#define IP_DF 0x4000        /* dont fragment flag */
#define IP_MF 0x2022        /* more fragments flag */
#define IP_OFFMASK 0x1fff   /* mask for fragmentation bits */
   u_char  ip_ttl;          /* time to lve */
   u_char  ip_p;            /* protocol */
   u_short ip_sum;          /* checksum */
   struct  in_addr  ip_src, ip_dst;   /* src, dest address */
};
```

Funções do IP



» **`ipintr`**
  – Retira datagrama da fila, verifica cabeçalho
  – Verifica se datagrama chegou ao destino final ➔ se não, chama `ip_forward`
  – Desmutiplexa → chama, sincronamente, **input** do TCP ou UDP

» **`ip_forward`**
  – Encaminha pacote para outra interface

» **`ip_output`**
  – Constroi cabeçalho IP, se este ainda não existe
  – Determina encaminhamento