Key point

试卷题型：填空题（12 空）、判断题 5 题、简答题 5 题、综合题 3 题（其中一个是程序注释题）

有几个知识点提醒大家掌握：

1. socket 实现（掌握会调用哪些函数分别创建以下哪个结构：file、socket、inpcb、tcpcb 等）（课堂练习）

    falloc（15.6）、socreate、in_pcballoc(22.1)、tcp_newtcpcb（25.8）

2. udp_input 和 udp_output 函数

3. 满足哪些条件才会发送 ICMP 重定向报文

```
923  #define satosin(sa) ((struct sockaddr_in *)(sa))
924      if (rt->rt_ifp == m->m_pkthdr.rcvif &&
925          (rt->rt_flags & (RTF_DYNAMIC | RTF_MODIFIED)) == 0 &&
926          satosin(rt_key(rt))->sin_addr.s_addr != 0 &&
927          ipsendredirects && !srcrt) {
```

首先，只有在同一接口**(rt_ifp 和 rcvif)**上接收或重发分组时，才能应用重定向。

其次，被选择的路由本身必须没有被 **ICMP** 重定向报文创建或修改过 **(RTF_DYNAMIC|RTF_MODIFIED)**

而且该路由也不能是到默认目的地的**(0.0.0.0)**

全局整数 **ipsendredirects** 指定系统是否被授权发送重定向.

当传给 **ip_forward** 的参数 **srcrt** 指明系统是对分组路由选择的源时，禁止系统重定向

First, redirects are applicable only when a packet is received and resent on the same interface (rt_ifp and rcvif).

Next, the selected route must not have been itself created or modified by an ICMP redirect message (RTF_DYNAMIC | RTF_MODIFIED).

Third, the route cannot be to the default destination (0.0.0.0).

Forth, system need to have administrative authority to send redirects, which means the global integer ipsendredirects is 1.

Five, redirects are not suppressed when the system is source routing a packet as indicated by the srcrt argument passed to ip_forward, which means the srcrt argument is False.

4. 理解图 24.17 和图 24.18, 理解 TCP 的流量控制机制和收发窗口如何滑动，理解每个序号变量的含义

5. 在 IP 处理过程中可能会发生哪些差错，并发送哪些 icmp 差错报文　（作业中有,需要查看 ch8 ch10 中的代码）

    1、If ip_ttl has reached 1 (IPTTLDEC), an ICMP time exceeded message is returned to the sender and the packet is discarded. Otherwise, ip_forward decrements ip_ttl by IPTTLDEC.

    2、If no route can be found for the destination, an ICMP host unreachable error is returned and the packed discarded.

    3、If fragmentation is prohibited by the DF bit, ip_output discards the packet and returns EMSGSIZE. If the datagram was generated on this host, a transport protocol passes the error back to the process, but if the datagram is being forwarded, ip_forward generates an ICMP destination unreachable error with an indication that the packet could not be forwarded without fragmentation

    P221

    4、an ICMP time exceeded error be sent to the source host if the timer expires and the first fragment of the datagram has been received. P232

Error code from ip_output	ICMP message generated	Description
EMSGSIZE	ICMP_UNREACH_NEEDFRAG	The outgoing packet was too large for the selected interface and fragmentation was prohibited (Chapter 10).
ENOBUFS	ICMP_SOURCEQUENCH	The interface queue is full or the kernel is running short of free memory. This message is an indication to the source host to lower the data rate.
EHOSTUNREACH ENETDOWN EHOSTDOWN default	ICMP_UNREACH_HOST	A route to the host could not be found. The outgoing interface specified by the route is not operating. The interface could not send the packet to the selected host. Any unrecognized error is reported as an ICMP_UNREACH_HOST error.

复习课

Chapter1

Net3 代码的组织层次

Chapter2

1、mbuf 结构体的字段和其含义

2、mdevget()函数：创建 mbuf 链表，根据数据大小四种情况

3、mget()函数，P32，分配 mbuf

4、mbuf 放分组的标志： M_PKTHDR

Chapter3

1、ifnet:通用接口结构

2、ifaddr：通用地址结构

3、le_softc ： 以太网的专用接口结构，arpcom ： 通用以太网结构

4、P61， main 函数，cpustartup 的功能

Chapter4

1、P82，ether_input()函数 ，作用，读懂

2、P84，ether_output()函数：验证、特定协议处理、构造帧、接口排队。调用 arpresolve 获得下一跳硬件地址

P85 line93，成功的话跳到 P87，放到发送缓存发送。不成功 return

Chapter6

P128 struct in_ifaddr 干什么的，是 ip 协议专用地址结构

P126 图 全局变量 in_ifaddr 指向全部 ip 地址，ifnet_addrs 指向数据链路层地址(MAC)

数据链路层地址：ifaddr+2 个 sockaddrdl？（没听清）

P128 图 6-8、6-9 看懂

Chapter7

P153 domaininit()作用

P158 ip_protox 数组是干嘛的、ipinit()函数作用

加框题目：复习课提到

加问号答案：不清楚，找不到明确答案

Chapter 1. Introduction

6. Which communication protocol families do 4.4BSD support?

    1、TCP/IP

    2、XNS (Xerox Network Systems)

    3、The OSI protocols

    4、The Unix domain protocols.

7. Two popular application programming interfaces (APIs) for writing programs to use the Internet protocols are **socket** and TLI (Transport Layer Interface).

8. The networking code in the 4.4BSD kernel is organized into three layers: **socket layer, protocol layer, interface layer.** The **interface** layer contains the device drivers that communicate with the network devices.

9. Data structures that are created by the kernel when the process calls socket are: **filedesc, socket, vnode.**

10. The return value from socket is a descriptor. The socket system calls start with a descriptor. Please describe how a descriptor leads to a socket structure.
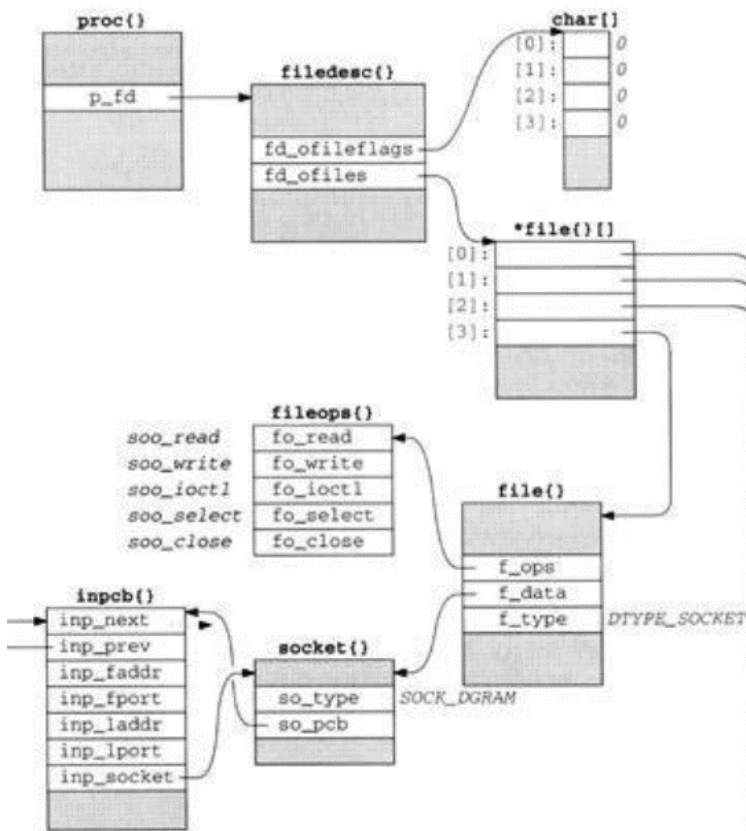
When the process executes a system call, the kernel starts with the descriptor value and uses "fd_ofiles" to index into the vector of "file" structure pointers, ending up with the "file" structure for the descriptor. The "file" structure points to "socket" structure.

11. When a UDP datagram(数据包) arrives on a network interface, how does the kernel find the corresponding socket structure?

    The kernel searches through all the UDP protocol control blocks to find the appropriate one, minimally based on the destination UDP port number and perhaps the destination IP address, source IP address, and source port numbers too. Once the inpcb structure is located, the kernel finds the corresponding socket structure through the inp_socket pointer

12. In the call to sendto, the fifth argument points to an Internet socket address structure (named serv) and the sixth argument specifies its length (which is 16 bytes). One of the first things done by the socket layer for this system call is to verify that these arguments are valid (i.e., the pointer points to a piece of memory in the address space of the process) and then copy the socket address structure into an mbuf. What is the m_type member of the resulting mbuf ?   or   What is the m_type member of the mbuf containing socket address structure?

    The "m_type" member specifies the type of data contained in the mbuf, which for this example is MT_SONAME, name of the socket.

13. What is the protocol layer corresponding to a UDP socket descriptor?

A UDP socket

14. For the call to sendto, the socket layer copies the destination socket address structure into an mbuf (Figure 1.6) and the data into an mbuf chain (Figure 1.7). After that, the protocol layer corresponding to the socket descriptor (a UDP socket) is called. Which routine is called?

    The protocol layer corresponding to the socket descriptor (a UDP socket) is called.

15. What does the UDP output routine mainly do?

    Specifically, the UDP output routine is called and pointers to the mbufs that we've examined are passed as arguments. This routine needs to prepend an IP header and a UDP header in front of the 150 bytes of data, fill in the headers, and pass the mbufs to the IP output routine.

16. For the call to sendto in the example program, the kernel has made three complete passes of the user data. Please describe the detail

    (1)copy the data from the user's buffer into the kernel's mbuf;
    (2)calculate the UDP checksum;
    (3)copy the data to it's transmit buffer and initiates the output

17. Give an overview of the processing that takes place when a process calls sendto to transmit a single UDP datagram.
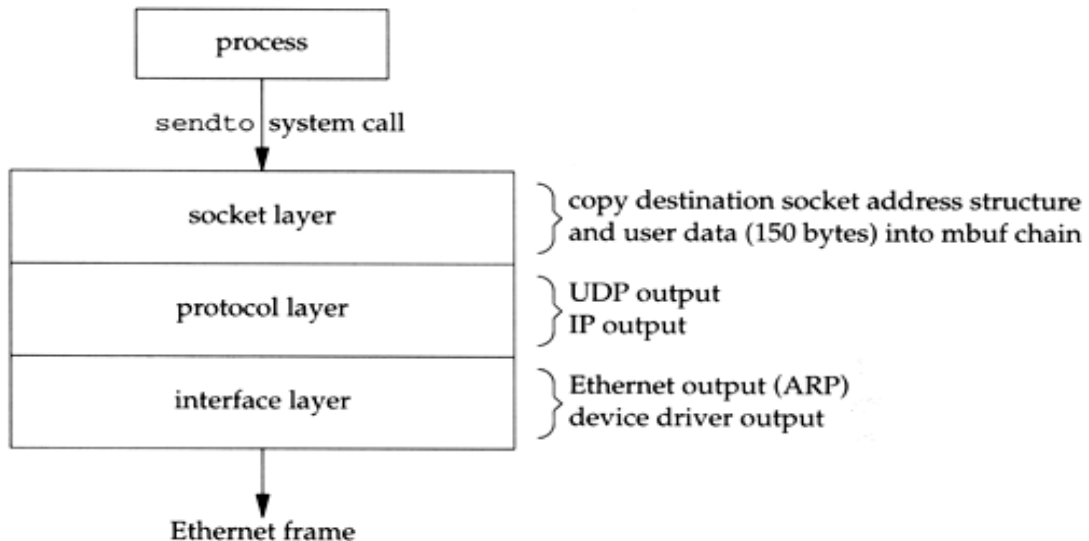
18. In chapter 1, the textbook gives an example program: send a datagramto the UDP daytime server and read a response. The program calls recvfrom to read the server's reply, after it calls sendto to send a datagramto the daytime server. If the reply has not been received yet, the input queue for the specified socket is empty. What is the state of the process at this point (the program calls recvfrom)?

    The process is put to sleep. (Sleeping)

19. Is input processing synchronous or asynchronous?

    Aynchronous

20. Is the reception of an input packet triggered by a receive-complete interrupt to a network device driver, or by a system call issued by the process?

    A receive-complete interrupt to a network device driver

21. What does the Ethernet Input do in the 4.4BSD-Lite distribution?

    It looks at the type field in the Ethernet frame to determine which protocol layer should receive the packet. In this example, the type field will specify an IP datagram, causing the mbuf to be added to the IP

input queue..

22. The IP software interrupt is set by **interface layer** (socket layer/protocol layer/interface layer)

23. What does the IP Input do in the 4.4BSD-Lite distribution?

 The IP input routine processes each IP datagram that it receives. It verifies the IP header checksum, processes any IP options, verifies that the datagram was delivered to the right host (by comparing the destination IP address of the datagram with the host's IP addresses), and forwards the datagram if the system was configured as a router and the datagram is destined for some other IP address. If the IP datagram has reached its final destination, the protocol field in the IP header specifies which protocol's input routine is called: ICMP, IGMP, TCP, or UDP. In our example, the UDP input routine is called to process the UDP datagram.

24. What does the UDP input do in the 4.4BSD-Lite distribution?

 The UDP input routine verifies the fields in the UDP header (the length and optional checksum) and then determines whether or not a process should receive the datagram.

25. In the example program we never specify the local port number for the application. We'll see in Exercise 23.3 that a side effect of writing the first UDP datagram(数据包) to a socket that has not yet bound a local port number is the automatic assignment by the kernel of a local port number (termed an ephemera(短暂的)l port) to that socket. That's how the inp_lport member of the PCB for our socket gets set to some nonzero value. Please find the corresponding codes in in_pcbconnect function, and in_pcbbind function. [可以学完 23 章再做]

```
                                                                  —— in_pcb.c
113    if (lport == 0)
114        do {
115            if (head->inp_lport++ < IPPORT_RESERVED ||
116                head->inp_lport > IPPORT_USERRESERVED)
117                head->inp_lport = IPPORT_RESERVED;
118            lport = htons(head->inp_lport);
119        } while (in_pcblookup(head,
120                        zeroin_addr, 0, inp->inp_laddr, lport, wild));
121    inp->inp_lport = lport;
122    return (0);
123 }
                                                                  —— in_pcb.c
```

图22-23 in_pcbbind 函数：选择一个临时端口

```
                                                                  —— in_pcb.c
227    if (in_pcblookup(inp->inp_head,
228                    sin->sin_addr,
229                    sin->sin_port,
230            inp->inp_laddr.s_addr ? inp->inp_laddr : ifaddr->sin_addr,
231                    inp->inp_lport,
232                    0))
233        return (EADDRINUSE);

234    if (inp->inp_laddr.s_addr == INADDR_ANY) {
235        if (inp->inp_lport == 0)
236            (void) in_pcbbind(inp, (struct mbuf *) 0);
237        inp->inp_laddr = ifaddr->sin_addr;
238    }
239    inp->inp_faddr = sin->sin_addr;
240    inp->inp_fport = sin->sin_port;
241    return (0);
242 }
                                                                  —— in_pcb.c
```

图22-28 in_pcbconnect 函数：验证插口对是唯一的

26. What does the example process input do?

 The program reads the datagram(数据包) that the server sends back by calling recvfrom. Our

program overwrites the carriage return with a null byte and calls printf to output the result.

27. Which queues are used for communication between the layers for network input and output?

    Socket queues and interface interface queues
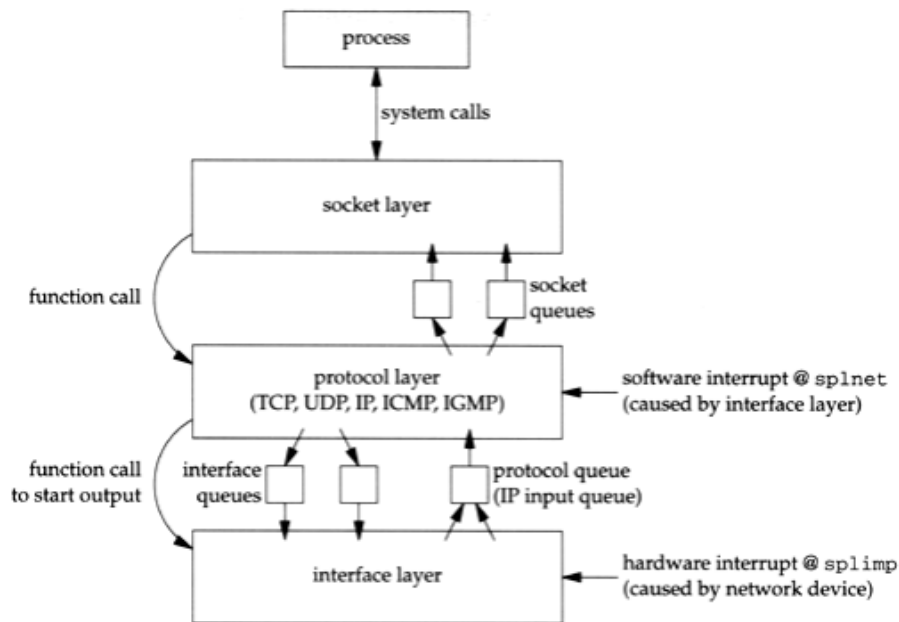
28. Where should a process, if blocked in its call to recvfrom, be stored in a socket structure, so as to be notified?

    struct socket
    {
    **   …**
    **   struct   sockbuf**
    **   {**
    **   …**
    **   struct   selinfo sb_sel;**
    **   …**
    **   }**
    **   …**
    }

    struct selinfo {
    **   pid_t   si_pid;   /* process to be notified */**
    **   short   si_flags; /* see below */**
    };

Chapter 2. Mbufs: Memory Buffers

1. What is the use of mbufs?

    The main use of mbufs is to hold the user data that travels from the process to the network interface, and vice versa. But mbufs are also used to contain a variety of other miscellaneous data: source and destination addresses, socket options, and so on.

2. In each of the mbufs, the m_data member points to the beginning of the corresponding buffer (either the mbuf buffer

itself or **external cluster**). Can this pointer point anywhere in the corresponding buffer, not necessarily the front? **YES**

3. Mbufs with a cluster always contain the starting address of the buffer (**m_ext.ext_buf)** and its size (m_ext.ext_size).

4. The **m _next** pointer links together the mbufs forming a single packet (record) into an mbuf chain, where the first mbuf has an m_flags value of **M_PKTHDR**, specifying a packet header.

5. The **m_nextpkt** pointer links multiple packets (records) together to form a queue of mbufs.

6. Write the content of an mbuf if its mh_flags is set as M_PKTHDR.
```
struct mbuf {
    struct    m_hdr m_hdr;
    union {
        struct {
            struct    pkthdr MH_pkthdr;    /* M_PKTHDR set */
            union {
                char      MH_databuf[MHLEN];
            } MH_dat;
        } MH;
    } M_dat;
};
```

7. The function that allocates an mbuf is **m_get** .

8. When an Ethernet frame is received, the device driver calls the function **m_devget** to create an mbuf chain and copy the frame from the device into the chain.

9. m_devget creates an mbuf chain. If the amount of data is greater than or equal to **208 bytes** (MINCLBYTES), one or more clusters are used.

10. The macro **mtod** returns a pointer to the data associated with an mbuf, and casts the pointer to a specified type.

11. The macro **dtom** takes a pointer to data anywhere within the data portion of the mbuf and returns a pointer to the mbuf structure itself.

12. When one of the protocols (IP, ICMP, IGMP, UDP, or TCP) finds that the amount of data in the first mbuf (m_len) is less than the size of the minimum protocol header (e.g., 20 for IP, 8 for UDP, 20 for TCP). m_pullup is called on the assumption that the remaining part of the header is in the next mbuf on the chain. The code is as below:
```
if (m->m_len < sizeof(struct ip) &&
    (m = m_pullup(m, sizeof(struct ip))) == 0)   {
        ipstat.ips_toosmall++;
        goto next;
}
ip = mtod(m, struct ip *);
```

What does the m_pullup do?

m_pullup is called on the assumption that the remaining part of the header is in the next mbuf on the chain. m_pullup rearranges the mbuf chain so that the first N bytes of data are contiguous（连续的） in the first mbuf on the chain. N is an argument to the function that must be less than or equal to 100 (MHLEN). If the first N bytes are contiguous（连续的） in the first mbuf, then both of the macros mtod and dtom will work.

13. m_pullup is called for every received IP fragment, when the IP fragment is stored in a cluster. This means that m_pullup is called for almost every received fragment, since **the length of most fragments is greater than 208 bytes** .

14. If the data is contained in a cluster, does the m_copy function do a physical copy of the data?
    No, an additional reference is made to the cluster instead.

15. In Section 2.6 two reasons that m_pullup can fail are listed. There are really three reasons. Obtain the source code (4.4BSD-Lite\usr\src\sys\kern\uipc_mbuf.c) for this function and discover the additional reason. Please list the three reasons.
    (1) if it needs another mbuf and its call to MGET fails.
    (2) if the total amount of data in the mbuf chain is less than the requested number of contiguous（连续的） bytes. The second reason is the most common cause of failure.
    (3) The caller asks for more than 100 (MHLEN) contiguous（连续的） bytes.

16. To avoid the problems we described in Section 2.6 with the dtom macro when the data is in a cluster, why not just add a back pointer to the mbuf for each cluster?
    This is infeasible（不可行的） since clusters can be pointed to by multiple mbufs (Figure 2.26 in Section 2.9). Also, there is no room in a cluster for a back pointer.

17. Since the size of an mbuf cluster is a power of 2 (typically 1024 or 2048), space cannot be taken within the cluster for the reference count. Obtain the Net/3 sources (Appendix B) and determine where these reference counts are stored.
    In the macros MCLALLOC and MCLFREE in <sys/mbuf.h> we see that the reference count is an array named mclrefcnt. This array is allocated when the kernel is initialized in the file machdep.c.

Chapter 3 . Interface Layer

1. When the ifnet and ifaddr structures are allocated for each network interface?
   During system initialization.

2. The ifnet structures are linked into the **ifnet** list.

3. **cpu_startup** locates and initializes all the hardware devices connected to the system, including any network interfaces.

4. Each device driver for a network interface initializes a specialized ifnet structure and calls if_attach to insert the structure into the linked list of interfaces. The **le_softc** structure is the specialized ifnet structure for the LANCE Ethernet device driver from Net/3. The **arpcom** structure is common to all Ethernet drivers.

5. The kernel constructs（构造） the link-level address by allocating memory for an ifaddr structure and two **sockaddr_dl** structures—one for the link-level address itself and one for the link-level address mask.

6. Each time if_attach is called it adds another ifnet structure to the **ifnet** list, creates a link-level ifaddr structure for the interface (which contains two **sockaddr_dl** structures), and initializes an entry in the **ifnet_addrs** array.

Chapter 4. Interfaces: Ethernet

1. In leread, **m_devget** copies the data from the buffer passed to leread to an mbuf chain it allocates.

2. ether_input jumps according to the Ethernet type field. For an IP packet, schednetisr schedules **an IP software interrupt** and the IP input queue, **ipintrq**, is selected (Please write the corresponding codes). For an ARP packet, the ARP software interrupt is scheduled and **arpintrq** is selected.   (code: next page)

3. In ether_output, the AF_INET case calls **arpresolve** to determine the Ethernet address corresponding to the destination IP address. If the Ethernet address is already in the ARP cache, **arpresolve** returns 1 and ether_output proceeds. Otherwise this IP packet is held by ARP, and when ARP determines the address, it calls ether_output from the function in_arpinput.

```
switch (eh->ether_type){
    case ETHETYPE_IP:
        schednetisr(NETISR_IP);
        inq = &ipintrq;
        break;
    case ETHERTYPE_ARP:
        schednetisr(NETISR_ARP);
        inq = &arpintrq;
        break;
}
```

4. Please explain the following codes in ether_output:
```
    M_PREPEND(m, sizeof (struct ether_header), M_DONTWAIT);
    if (m == 0)    senderr(ENOBUFS);
    eh = mtod(m, struct ether_header *);
    type = htons((u_short)type);
    bcopy((caddr_t)&type,(caddr_t)&eh->ether_type,
        sizeof(eh->ether_type));
    bcopy((caddr_t)edst, (caddr_t)eh->ether_dhost, sizeof (edst));
    bcopy((caddr_t)ac->ac_enaddr, (caddr_t)eh->ether_shost,
        sizeof(eh->ether_shost));
    s = splimp();
    if (IF_QFULL(&ifp->if_snd)) {
        IF_DROP(&ifp->if_snd);
        splx(s);
        senderr(ENOBUFS);
    }
    IF_ENQUEUE(&ifp->if_snd, m);
    if ((ifp->if_flags & IFF_OACTIVE) == 0)
        (*ifp->if_start)(ifp);
    splx(s);
```

Ethernet frame construction
152-167
If the code in the switch made a copy of the packet, the copy is processed as if it had been received on the output interface by calling looutput.

At this point, the mbuf contains a complete Ethernet frame except for the 32-bit CRC, which is computed by the Ethernet hardware during transmission. T

output queueing.

168-185

If the output queue is full, ether_output discards the frame and returns ENOBUFS. If the output queue is not full, the frame is placed on the interface's send queue, and the interface's if_start function transmits the next frame if the interface is not already active.

186-190

The senderr macro jumps to bad where the frame is discarded and an error code is returned.

Chapter 6. IP Addressing

1. Interface address structures and protocol address structures that are specialized for IP: the **in_ifaddr** and **sockaddr_in structures**.

2. in_ifaddr starts with the generic interface address structure **ia_ifa**, followed by the IP-specific members.

3. For each IP address assigned to an interface, an in_ifaddr structure is allocated and added to the interface address list **ia_next** and to the global list of IP addresses **in_ifaddr** . all the link-level addresses can be accessed from the **ifnet_addrs** array.

4. Why do you think sin_addr in the sockaddr_in structure was originally defined as a structure?

Before IP subnetting (RFC 950 [Mogul and Postel 1985]), the network and host portions of IP addresses always appeared on byte boundaries. The definition of an in_addr structure was

```
struct in_addr {
        union {
                struct { u_char s_b1, s_b2, s_b3, s_b4; } S_un_b;
                struct { u_short s_w1, s_w2; } S_un_w;
                u_long S_addr;
        } S_un;
#define s_addr   S_un.S_addr        /* should be used for all code */
#define s_host   S_un.S_un_b.s_b2   /* OBSOLETE: host on imp */
#define s_net    S_un.S_un_b.s_b1   /* OBSOLETE: network */
#define s_imp    S_un.S_un_w.s_w2   /* OBSOLETE: imp */
#define s_impno  S_un.S_un_b.s_b4   /* OBSOLETE: imp # */
#define s_lh     S_un.S_un_b.s_b3   /* OBSOLETE: logical host */
};
```

The Internet address could be accessed as 8-bit bytes, 16-bit words, or a single 32-bit address. The macros s_host, s_net, s_imp, and so on have names that correspond to the physical structure of early TCP/IP networks.

The use of subnetting and supernetting makes the byte and word divisions obsolete.

5. Why is the IP address duplicated in ac_ipaddr when it is already contained in an ifaddr structure on the interface's address list?

The interface output functions, such as ether_output, have a pointer only to the ifnet structure for the interface, and not to an ifaddr structure. Using the IP address in the arpcom structure (which is the last IP address assigned to the interface) avoids having to select an address from the ifaddr address list.

Chapter 7. Domains and Protocols

1. Net/3 groups related protocols into a **domain**, and identifies each domain with a **protocol family constant**.

2. At compile time, Net/3 allocates and initializes a **protosw** structure for each protocol in the kernel and groups the structures for all protocols within a single domain into an array.

3. A kernel may provide multiple interfaces to the same protocol by providing multiple protosw entries. Which protocol has 3 protosw structures in the inetsw array within the Net/3 kernel ?

IP protocol (inetsw[0], inetsw[3] and inetsw[6])?

4. The **pffindproto** and **pffindtype** functions are called to locate the appropriate protosw entry when a process creates a socket.

5. At system initialization time **domaininit** links the domains into the domains list, calls the domain and protocol initialization functions, and calls the fast and slow timeout functions.

6. What call to the pffindproto returns a pointer to inetsw[6] ?
 pffindproto(PF_INET, 27, SOCK_RAW); (below Fig. 7.20)

7. What call to the pffindproto returns a pointer to inetsw[1] ?
 socket(PF_INET, SOCK_STREAM, 0);   ->   pffindtype(PF_INET, SOCK_DGRAM);

8. What call to the pffindtype returns a pointer to inetsw[2] ?
 socket(PF_INET, SOCK_DGRAM, 0);   ->   pffindtype(PF_INET, SOCK_STREAM);

9. Which protosw structure in the inetsw array does pffindtype(PF_INET, SOCK_RAW) return a pointer to ?
 Inetsw[6]?

10. ip_init initializes the IP reassembly（再装配） queue **ipq**, seeds **ip_id** from the system clock, and sets the maximum size of the IP input queue (**ipintrq**) to 50 (ipqmaxlen).

11. IP must demultiplex incoming datagrams and deliver them to the appropriate transport-level protocols. This is done by the **ip_protox** array. The array maps the **protocol number** to an entry in the inetsw array.

Chapter 8. IP: Internet Protocol

1. The standard IP header is **20** bytes long, but may be followed by up to **40** bytes of options. IP can split large datagrams into **fragments** to be transmitted and reassembles the **fragments** at **the final destination**.

2. If the system is configured as a **unfinal destination** , datagrams that have not reached their final destination are sent to ip_forward for routing toward their final destination. （依据 P174 8.5）

3. If ip_src has not been specified, then ip_output selects ia, the **IP address of the outgoing interface**, as the source address.

4. Explains when the array ip_protox is initialized, and its function, and writes the codes using ip_protox n ipintr.
 When the ip_init function is called by domaininit) at system initialization time, the array ip_protox is initialized

5. Should IP accept broadcast packets when there are no IP addresses assigned to any interfaces?
 Probably not. The system could not respond to any broadcasts since it would have no source address to use in the reply.

6. Why isn't an error message returned to the sender when an IP packet arrives with checksum errors?
 Since the packet has been damaged, there is no way of knowing if the addresses in the header are correct or not.

7. Assume that a process on a multihomed host has selected an explicit source address for its outgoing packets. Furthermore, assume that the packet's destination is reached through an interface other than the one selected as the packet's source address. What happens when the first-hop router discovers that the packets should be going through a

different router? Is a redirect message sent to the host?

If an application selects a source address that differs from the address of the selected outgoing interface, redirects from the selected next-hop router fail. The next-hop router sees a source address different from that of the subnetwork on which it was transmitted and does not send a redirect message. This is a consequence of implementing the weak end system model and is noted in RFC 1122.

8. Why is it necessary to decrement ip_ttl after testing it (versus before) in Figure 8.17?

The decrement of the TTL is done after the comparison for less than or equal to 1 to avoid the potential error of decrementing a received TTL of 0 to become 255.

9. What would happen if two routers each considered the other the best next-hop destination for a packet?

If two routers each consider the other the best next-hop for a packet, a routing loop exists. Until the loop is removed, the original packet bounces between the two routers and each one sends an ICMP redirect back to the source host if that host is on the same network as the routers. Loops may exist when the routing tables are temporarily inconsistent during a routing update.

The TTL of the original packet eventually reaches 0 and the packet is discarded. This is one of the primary reasons why the TTL field exists.

10. ip_forward converts the fragment id from host byte order to network byte order before calling icmp_error. Why does it not also convert the fragment offset?

ICMP error messages are generated only for the initial fragment of a datagram(数据包), which always has an offset of 0. The host and network forms for 0 are the same, so no conversion is necessary.

Chapter 10. IP Fragmentation and Reassembly（再装配）

1. ip_output splits an outgoing datagram into fragments if it is too large to be transmitted on the selected network. Write the if statement.

```
/*
 * If small enough for interface, can just send directly.
 */
if ((u_short)ip->ip_len <= ifp->if_mtu) {
    ip->ip_len = htons((u_short)ip->ip_len);
    ip->ip_off = htons((u_short)ip->ip_off);
    ip->ip_sum = 0;
    ip->ip_sum = in_cksum(m, hlen);
    error = (*ifp->if_output)(ifp, m,
            (struct sockaddr *)dst, ro->ro_rt);
    goto done;
}
```

2. Why only the destination host can reassemble the original datagram(数据包)?

Because only the destination host has a chance to see all the fragments.

3. How to determine fragment size (the number of data bytes in each fragment)?

```
/*
 * Too large for interface; fragment if possible.
```

```
     * Must be able to put at least 8 bytes per fragment.
    */
   if (ip->ip_off & IP_DF) {
       error = EMSGSIZE;
       ipstat.ips_cantfrag++;
       goto bad;
   }
   len = (ifp->if_mtu - hlen) &~ 7;
   if (len < 8) {
       error = EMSGSIZE;
       goto bad;
   }
```

4. **ip_reass** accepts incoming fragments and attempts to reassemble datagram(数据包)s. If it is successful, the datagramis passed back to ipintr and then to the appropriate transport protocol. **ip_slowtimo** discards incomplete datagrams when all their fragments haven't been received within a reasonable amount of time.

Chapter 11. ICMP: Internet Control Message Protocol

1. What errors or exceptions may be encountered in the IP processing and what ICMP error messages may be sent? (Answer in Chinese) the same as Exercise 5 in Chapter 10.

   1.ip_forward 可能会由于 ip_output 失败或重定向而发送 ICMP 报文（书 P180 5）
   2.当收到数据报的第一个分片且定时器超时时，向源主机发送一个 ICMP 超时差错报文
   3.当路由器收到一个需要分片的数据报，而 IP 首部'DF'位置 1 时，发送 ICMP 不可达差错报文

2. If the ICMP information request is obsolete, why does icmp_input pass it to rip_input instead of discarding it?

   By passing the message to rip_input, a process-level daemon could respond and old systems that relied on this behavior could continue to be supported.

3. We pointed out that Net/3 does not convert the offset and length field of an IP packet to network byte order before including the packet in an ICMP error message. Why is this inconsequential in the case of the IP offset field?

   ICMP errors are sent only for the initial fragment of an IP datagram. Since the offset value of an initial fragment is always 0, the byte ordering of the field is unimportant.

4. In Figure 11.30, what happens to a packet that has the high-order bit of ip_off set?

   The high-order bit is reserved and must be 0. If it is sent, icmp_error will discard the packet.

5. Why is the return value from ip_output discarded in Figure 11.39?

   The return value is discarded because icmp_send does not return an error, but more significantly, errors generated during ICMP processing are discarded to avoid generating an endless series of error messages.