协议工程上机3

闫一慧 20009200331

tcp.h

该代码是 Berkeley Software Distribution (BSD) 操作系统中传输控制协议 (TCP) 的头文件。头文件定义了与 TCP 相关的各种常量、结构和选项。

以下是有关代码的一些要点：

- 代码定义了 TCP 报头结构（`struct tcphdr`）和各种 TCP 选项。
- 它包括 TCP 标志（`TH_FIN`、`TH_SYN` 等）及其对应的十六进制值的定义。
- 该代码还包括最大段大小 (MSS)、窗口大小、选项长度和其他 TCP 相关参数的常量。
- 定义了用户可设置的选项，如TCP_NODELAY、TCP_MAXSEG、TCP_NOPUSH等，可与setsockopt函数一起使用来配置TCP套接字选项。
- 该代码包括一些与 TCP 性能、日志记录、拥塞控制和保活功能相关的附加选项和常量。

总的来说，这个头文件提供了在 BSD 操作系统中使用 TCP 连接所必需的定义和选项。

`tcphdr`结构定义了TCP首部:

```
struct tcphdr {
    u_short th_sport;       /* source port */
    u_short th_dport;       /* destination port */
    tcp_seq th_seq;         /* sequence number */
    tcp_seq th_ack;         /* acknowledgement number */
#if BYTE_ORDER == LITTLE_ENDIAN
    u_char  th_x2:4,        /* upper 4 (reserved) flags */
        th_off:4;       /* data offset */
#endif
#if BYTE_ORDER == BIG_ENDIAN
    u_char  th_off:4,       /* data offset */
        th_x2:4;        /* upper 4 (reserved) flags */
#endif
    u_char  th_flags;
#define TH_FIN  0x01
```

```
#define TH_SYN  0x02
#define TH_RST  0x04
#define TH_PUSH 0x08
#define TH_ACK  0x10
#define TH_URG  0x20
#define TH_ECE  0x40
#define TH_CWR  0x80
#define TH_AE   0x100           /* maps into th_x2 */
#define TH_RES3 0x200
#define TH_RES2 0x400
#define TH_RES1 0x800
#define TH_FLAGS    (TH_FIN|TH_SYN|TH_RST|TH_PUSH|TH_ACK|TH_URG|TH_ECE|TH_CWR)
#define PRINT_TH_FLAGS  "\20\1FIN\2SYN\3RST\4PUSH\5ACK\6URG\7ECE\10CWR\11AE"

    u_short th_win;         /* window */
    u_short th_sum;         /* checksum */
    u_short th_urp;         /* urgent pointer */
};
```

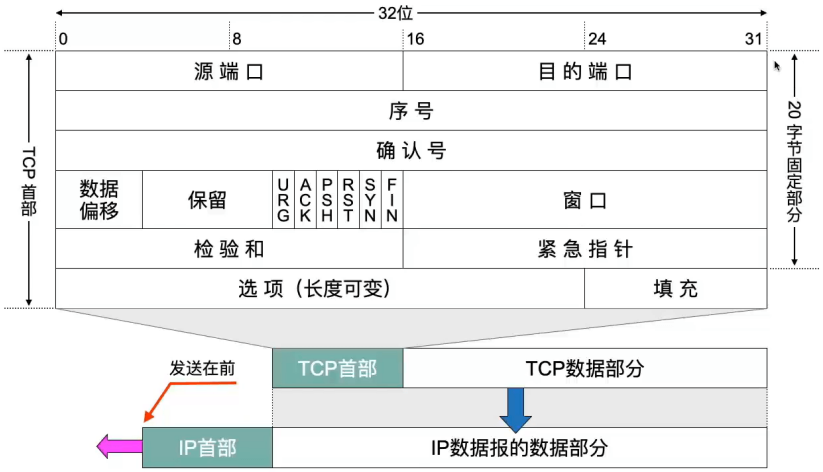
这段代码定义了一个名为 `tcphdr` 的结构体，该结构体表示TCP协议头部的格式,下面是 `tcphdr` 结构体中各个字段的详细介绍：

- `th_sport`：16位无符号整数，表示源端口号，即发送方使用的端口号。
- `th_dport`：16位无符号整数，表示目标端口号，即接收方使用的端口号。
- `th_seq`：32位整数，表示序列号，用于按序传输数据。
- `th_ack`：32位整数，表示确认号，用于确认已接收的数据。
- `th_x2` 和 `th_off`：这两个字段是4位的位字段（bit-field），用于表示TCP头部中的一些标志位和数据偏移量。具体的位分配取决于字节序（大端序或小端序）。
  - 如果使用的是小端序（`BYTE_ORDER == LITTLE_ENDIAN`），则 `th_x2` 占用头部的高4位，用于表示保留的标志位，`th_off` 占用头部的低4位，用于表示数据偏移量（即TCP头部的长度）。
  - 如果使用的是大端序（`BYTE_ORDER == BIG_ENDIAN`），则 `th_off` 占用头部的低4位，`th_x2` 占用头部的高4位。
- `th_flags`：8位无符号整数，表示TCP头部的标志位，用于控制TCP连接的状态和行为。具体的标志位有：
  - `TH_FIN`：表示发送端已经完成发送数据，请求关闭连接。
  - `TH_SYN`：表示请求建立一个连接。
  - `TH_RST`：表示重置连接。

- `TH_PUSH`：表示数据包立即传输，而不是等待缓冲区填满。
- `TH_ACK`：表示确认号有效。
- `TH_URG`：表示紧急指针字段有效。
- `TH_ECE`：表示ECN（Explicit Congestion Notification）被支持和使用。
- `TH_CWR`：表示拥塞窗口减小。
- `TH_AE`：表示由`th_x2`字段扩展，具体含义可根据上下文而定。
- `TH_RES3`、`TH_RES2`、`TH_RES1`：保留字段。

- `th_win`：16位无符号整数，表示窗口大小，用于流量控制。
- `th_sum`：16位无符号整数，表示校验和，用于检测数据在传输过程中是否发生错误。
- `th_urp`：16位无符号整数，表示紧急指针，指示紧急数据的结束位置。

此外，代码中还定义了一些宏和常量，用于表示TCP头部的各个标志位以及打印标志位的格式，通过使用该结构体，可以方便地处理TCP协议头部的各个字段和标志位，对TCP连接进行控制和管理。



tcp_input.c

`tcp_input.c`文件包含了TCP协议的输入处理函数，负责接收和处理TCP报文的逻辑。下面是对`tcp_input.c`文件中主要代码功能的一般介绍：

1. 接收TCP报文：
   - `tcp_input()`函数是TCP协议栈的入口函数，负责处理接收到的TCP报文。
   - 通过解析报文首部，提取源端口、目的端口、序列号、确认号、标志位、窗口大小等信息。
2. TCP连接的建立和终止：
   - 处理TCP的三次握手建立连接过程，包括处理SYN和ACK标志位、序列号和确认号的确认等。
   - 处理TCP的四次挥手终止连接过程，包括处理FIN和ACK标志位的确认、发送和接收最后的ACK等。

3. 数据传输和窗口管理：
    - 确保接收到的数据按序传递给应用程序，处理乱序到达的报文段。
    - 维护接收和发送窗口，进行流量控制和拥塞控制。
4. TCP报文的处理：
    - 处理TCP报文段的分片和重组。
    - 检查校验和，确保数据的完整性。
5. TCP状态管理和超时处理：
    - 根据接收到的TCP报文和当前的状态，进行状态转换和处理。
    - 处理超时事件，包括重传丢失的报文段和触发拥塞控制机制。

总体而言，`tcp_input.c`文件中的代码实现了TCP协议栈的核心功能，包括连接管理、数据传输、窗口控制、拥塞控制等。它是网络通信中的重要组成部分，负责处理和管理TCP连接的各种情况和事件,由于代码较长此处不进行展示。

以下是按照代码从上到下的顺序对其功能进行的总结：

1. 从第一个mbuf中获取IP和TCP首部
2. 验证TCP检验和
3. .验证TCP偏移字段
4. . 把IP和TCP首部及选项放入第一个mbuf
5. . 快速处理时间戳选项
6. 保存输入标志，把字段转换为主机字节序
7. 寻找Internet PCB
8. .丢弃报文段并生成RST
9. 丢弃报文段且不发送响应
10. 不改变通告窗口大小
11. 如果选定了插口调试选项，则保存连接状态及 I P和T C P首部
12. 如果监听插口收到了报文段，则创建新的插口
13. 计算窗口缩放因子
14. 复位空闲时间和保活定时器
15. 如果不处于监听状态，处理 T C P选项
16. 首部预测
17. 完成接收数据的处理

18. 首部预测失败时的处理代码
19. 完成被动打开或主动打开
20. PAWS：防止序号回绕
21. 裁剪报文段使数据在窗口内
22. 自连接和同时打开
23. 记录时间戳
24. RST处理
25. 完成被动打开和同时打开
26. 快速重传和快速恢复的算法
27. ACK处理
28. 更新窗口信息
29. 紧急方式处理
30. 处理已接收的数据
31. FIN处理
32. 利用复制数据计算检验和，并 避免在处理中多次遍历数据

以上就是对各段代码的功能总结，这段代码很长，实现的功能很多，且非常精妙，非常让人感叹。

tcp_output.c

tcp_output函数很大同样按照上面的方法给出大概的代码功能理解：

1. 对是否等待对端的A C K进行判断和确认
2. 返回慢启动
3. 发送多个报文段
4. 计算发送的数据量
5. 窗口缩小检查
6. 进入持续状态
7. 一次发送一个报文段
8. 如果发送缓存不空，关闭F I N标志
9. 计算接收窗口大小
10. 发送方避免糊涂窗口综合症的方法
11. 发送一个报文段

> t c p _ o u t p u t接下来的代码负责发送报文段—填充T C P报文首部的所有字段，并传递给I P层准备发送。

12. 构造M S S选项
13. 是否发送窗口大小选项
14. 构造窗口大小选项
15. 构造时间戳选项
16. 选项加入后是否会造成报文段长度越界
17. 更新统计值
18. 为I P和T C P首部分配m b u f
19. 置位P S H标志
20. 得到存储I P和T C P首部的m b u f
21. 向m b u f中复制I P和T C P首部模板
22. 设置报文段的序号字段
23. 设置报文段的确认字段
24. 设置紧急数据偏移量
25. 保存起始序号
26. 增加s n d _ n x ,更新s n d _ m a x
27. 设定重传定时器
28. 为插口调试添加路由记录
29. 设置I P长度、T T L和TO S
30. 向I P传递数据报
31. 确认是否还有数据需要发送

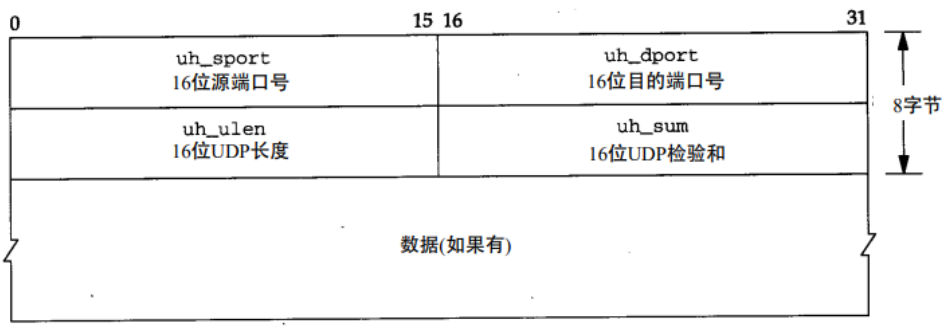
总体而言， `tcp_output.c` 文件中的代码实现了TCP协议栈的输出功能，包括构建和发送TCP报文、连接管理、数据传输、窗口控制、拥塞控制等。它负责将应用程序的数据封装为TCP报文，并通过网络传输到目标主机。同时，它还负责管理和维护TCP连接的各种状态和参数，以确保可靠的数据传输。

udp.h

这段代码定义了UDP协议头（`struct udphdr`）以及相关的常量和选项。下面是对代码的功能的介绍：

1. `#ifndef _NETINET_UDP_H_`：条件预处理指令，如果`_NETINET_UDP_H_`未定义，则执行下面的代码。这是为了防止重复包含头文件。
2. `#define _NETINET_UDP_H_`：定义`_NETINET_UDP_H_`，用于标记已经包含了该头文件。
3. `#include <sys/types.h>`：包含系统类型头文件，以便使用`u_short`等类型。
4. `struct udphdr`：定义UDP协议头的结构体。
   - `uh_sport`：源端口号。
   - `uh_dport`：目的端口号。
   - `uh_ulen`：UDP报文长度。
   - `uh_sum`：UDP校验和。
5. `#define UDP_ENCAP 1`：定义UDP封装选项的常量。该选项用于设置UDP封装。
6. `#define UDP_VENDOR SO_VENDOR`：定义第三方用户可设置选项的保留空间的起始位置。
7. `#define UDP_ENCAP_ESPINUDP_NON_IKE 1`：定义UDP封装的类型常量。表示非IKE的ESP在UDP封装。
8. `#define UDP_ENCAP_ESPINUDP 2`：定义UDP封装的类型常量。表示ESP在UDP封装。
9. `#define UDP_ENCAP_ESPINUDP_PORT 500`：定义ESP在UDP封装中的默认端口号。
10. `#define UDP_ENCAP_ESPINUDP_MAXFRAGLEN 552`：定义ESP在UDP封装中的最大UDP片段长度。
11. `#endif`：条件预处理指令的结束标记。

这段代码的主要功能是定义UDP协议头的结构体和相关常量，以及UDP封装选项的常量。它为使用UDP协议的应用程序提供了必要的数据结构和选项定义。通过包含此头文件，可以在应用程序中使用`struct udphdr`结构体来构建和解析UDP报文头部，以及使用UDP封装选项来设置UDP封装相关的参数，头文件示意图如下。



TCP与UDP总结与对比

通过学到的知识和阅读的代码了解到，TCP（Transmission Control Protocol）和UDP（User Datagram Protocol）是两种常用的传输层协议，用于在计算机网络中传输数据。它们有以下几个主要区别：

1. **可靠性：**
   - TCP是面向连接的协议，提供可靠的数据传输。它使用可靠的数据传输机制，通过序号、确认和重传来确保数据的完整性和可靠性。TCP还具有拥塞控制和流量控制机制，以防止网络拥塞和数据丢失。
   - UDP是无连接的协议，不提供可靠性保证。它仅提供数据报的传输，不进行确认和重传操作。因此，UDP更适用于那些对实时性要求较高、可以容忍少量数据丢失的应用，如音视频传输和实时游戏。
2. **连接性：**
   - TCP是面向连接的协议，通信之前需要先建立连接，然后进行数据传输，最后再释放连接。连接的建立和释放过程需要时间开销，但确保了数据的顺序和可靠性。
   - UDP是无连接的协议，通信时不需要建立连接，直接发送数据报。由于没有连接建立和断开的过程，UDP具有较低的延迟。
3. **数据报结构：**
   - TCP使用字节流方式进行数据传输，将数据划分为TCP报文段进行传输。TCP报文段包含序号、确认号、窗口大小等控制信息。
   - UDP使用数据报方式进行数据传输，每个UDP数据报都是独立的数据单元。UDP数据报由UDP头部和数据组成，UDP头部包含源端口、目的端口、长度和校验和等信息。
4. **流量控制和拥塞控制：**
   - TCP具有流量控制和拥塞控制机制，通过动态调整发送速率和维护拥塞窗口来适应网络的状况，以避免网络拥塞和数据丢失。
   - UDP没有内置的流量控制和拥塞控制机制，数据发送速率由应用程序控制，如果发送速率过快或网络拥塞，可能导致数据丢失。
5. **应用场景：**
   - TCP适用于对数据可靠性要求较高的应用，如网页浏览、文件传输、电子邮件等。
   - UDP适用于对实时性要求较高，能容忍少量数据丢失的应用，如音视频流媒体、实时游戏、DNS查询等。

总结来说，TCP提供可靠的、面向连接的数据传输，适用于对数据完整性和顺序性要求较高的场景；UDP提供简单的、无连接的数据传输，适用于对实时性要求较高、可以容忍少量数据丢失的场景。选择使用TCP还是UDP取决于具体应用的需求和特点。