Network Programming: Ch.7: Socket Options

Li-Hsing Yen
NYCU

Ver. 1.0.0

Socket Options

- *getsockopt* and *setsockopt* functions
- Check options and obtain default values
- Generic socket options
- IPv4 socket options
- IPv6 socket options
- TCP socket options
- *fcntl* function
- Summary

Some materials in these slides are taken from Prof. Ying-Dar Lin with his permission of usage

getsockopt and *setsockopt* Functions

```
#include <sys/socket.h>
int getsockopt (int sockfd, int level, int optname, void *optval,
    socklen_t *optlen);
int setsockopt (int sockfd, int level, int optname, const void *optval,
    socklen_t optlen);
                                        Both return: 0 if OK, -1 if error
```

Types of options: flag and value

Levels:

Generic --	SOL_SOCKET
IP --	IPPROTO_IP
ICMPv6 --	IPPROTO_ICMPV6
IPv6 --	IPPROTO_IPV6
TCP --	IPPROTO_TCP
SCTP --	IPPROTO_SCTP

Check Options and Obtain Default Values

(Program Outline: `sockopt/checkopts.c`)

- Program declaration:
  - declare union of possible option values
  - define printing function prototypes
  - define `sock_opts` structure, initialize `sock_opt[]` array
- Check and print options:
  - create TCP socket, go through all options
  - call *getsockopt*
  - print option's default value

Declaration for Socket Options (see Figure 7.3)

Program to Check and Print Socket Options (see Figs. 7.4 & 7.5)

```
[lhyen@nplinux1 sockopt]$ ./checkopts
SO_BROADCAST: default = off
SO_DEBUG: default = off
SO_DONTROUTE: default = off
SO_ERROR: default = 0
SO_KEEPALIVE: default = off
SO_LINGER: default = l_onoff = 0, l_linger = 0
SO_OOBINLINE: default = off
SO_RCVBUF: default = 87380
SO_SNDBUF: default = 16384
SO_RCVLOWAT: default = 1
SO_SNDLOWAT: default = 1
SO_RCVTIMEO: default = 0 sec, 0 usec
SO_SNDTIMEO: default = 0 sec, 0 usec
SO_REUSEADDR: default = off
```

```
SO_REUSEPORT: default = off
SO_TYPE: default = 1
IP_TOS: default = 0
IP_TTL: default = 64
IPV6_DONTFRAG: default = off
IPV6_UNICAST_HOPS: default = 64
IPV6_V6ONLY: default = off
TCP_MAXSEG: default = 536
TCP_NODELAY: default = off
SCTP_AUTOCLOSE: (undefined)
SCTP_MAXBURST: (undefined)
SCTP_MAXSEG: (undefined)
SCTP_NODELAY: (undefined)
[lhyen@nplinux1 sockopt]$
```

Generic Socket Options (handled within the kernel)

- SO_BROADCAST: permit sending of broadcast datagram (IP or UDP), only on broadcast links
- SO_DEBUG: enable the kernel to track details about packets sent or received by TCP socket, allowing *trpt* program to examine the kernel circular buffer
- SO_DONTROUTE: bypass routing table lookup, used by routing daemons (`routed` and `gated`) to force a packet to be sent out a particular interface

SO_ERROR Socket Option (1/2)

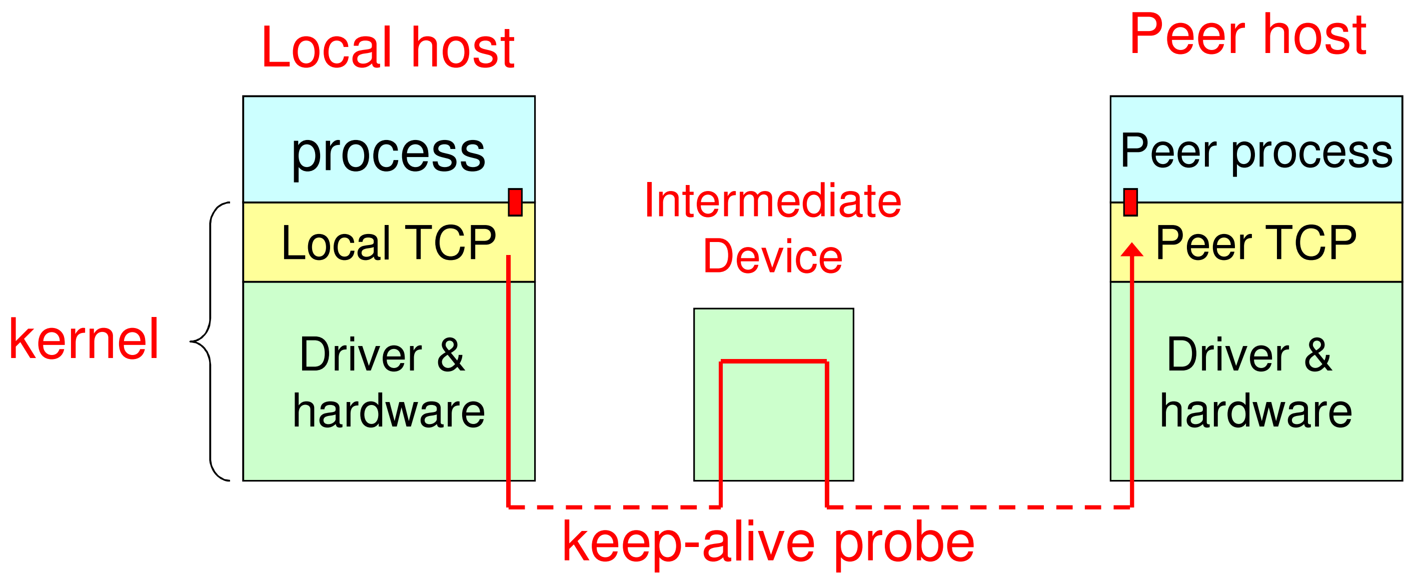
- SO_ERROR: get pending error and clear
- 當 socket 發生錯誤時，kernel 會將錯誤代碼放入變數 so_error中，稱為該 socket 的pending error
  - 如process正block在select中測試該socket是否readable或writable，則select會return
  - 如該process使用signal-driven I/O (五種I/O model中的第四種)，則kernel會產生SIGIO signal給process
- 無論是哪一種情形，process都可以透過SO_ERROR option取得 so_error的值

SO_ERROR Socket Option (2/2)

- so_error的值被取回後即會被kernel歸零
- 當process呼叫read時so_error的值不為0且沒有data可讀，則read return -1
  - 連線已關閉時傳回0，有data可讀時傳回讀取byte數，皆不會傳回-1。
- 當process呼叫write時so_error的值不為0，則 write return -1 (無論是否writable)
- 此時so_error的值會被設給全域變數errno後歸零。Process只須檢查errno的值即可。

`SO_KEEPALIVE` Socket Option (1/2)

- SO_KEEPALIVE: when enabled, test if TCP connection still alive periodically (default 2 hours, can be changed by TCP_KEEPALIVE)



SO_KEEPALIVE Socket Option (2/2)

- 當任一方超過兩小時未交換資料，kernel TCP會自動發keep-alive probe (a TCP segment)給Peer TCP
  - 如果Peer TCP回應ack，process不受任何影響。
  - 如果Peer TCP回應RST (Peer process已不存在)，則local TCP close此socket且設so_error = ECONNRESET
  - 如果沒收到peer的回應，so_error = ETIMEOUT
  - 如果收到ICMP unreachable的錯誤， so_error = EHOSTUNREACH

Peer Host Vs. Peer Process

- The purpose of keep-alive is to detect if the peer *host* (not the peer *process*) crashes
  - If peer process crashes, peer TCP sends an FIN
    - If local TCP sends another segment, peer TCP responds with a RST
    - If process sends yet another segment, local TCP sends local process a `SIGPIPE` signal
    - For other cases see Figure 7.6
- This option is normally used by servers to detect if client host crashes (avoids *half-open*)

Application-Layer Timeout Mechanism

- Some servers (notably FTP servers) provide an application timeout to detect inactive clients
- This is often a better method than kernel’s keep-alive scheme, since the application has complete control

`SO_LINGER` Socket Option

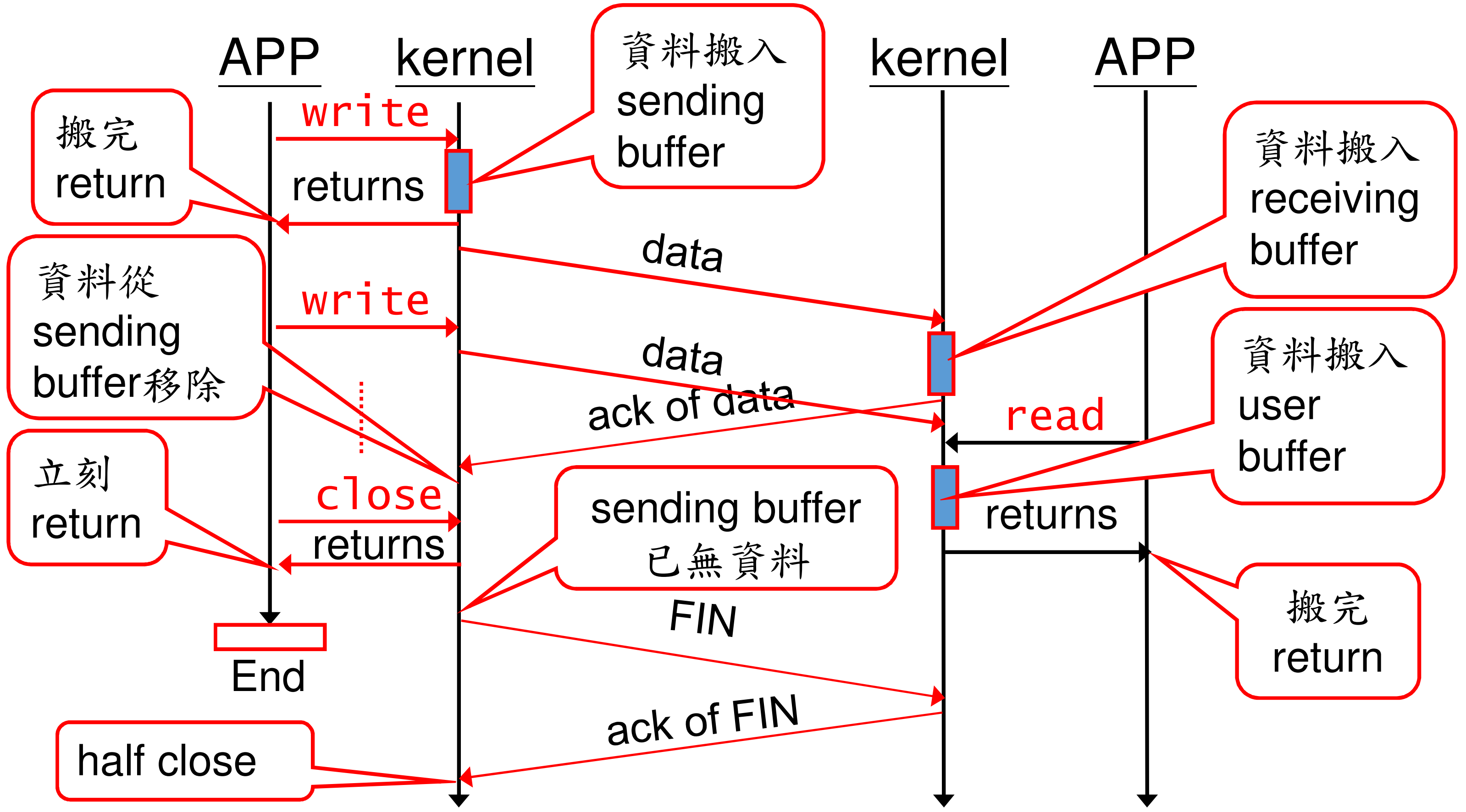
- Specifies how `close` operates for TCP/SCTP
- By default, `close` returns immediately
  - While the kernel will try to deliver any data remaining in the socket send buffer
- Application process may set this option to
  - enable TCP to abort the connection by discarding any pending data, or
  - put the process to sleep until (1) all the data is sent and acknowledged by the peer or (2) time out

SO_LINGER: Data Structure

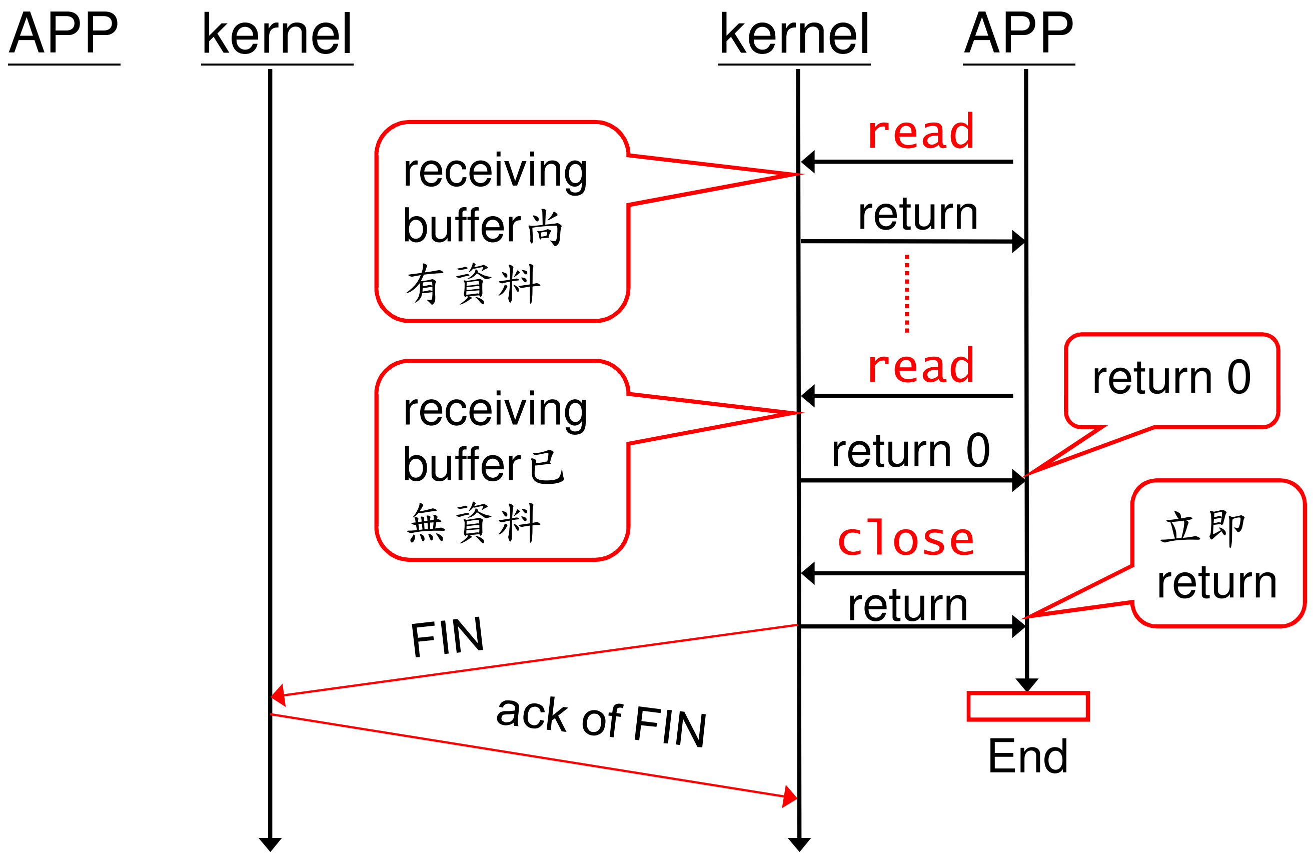
```
#include <sys/socket.h>

struct linger {
    int l_onoff;    /* 0=off, nonzero=on */
    int l_linger;   /* linger time */
};
```

① `l_onoff = 0: close` returns immediately (default)
② `l_onoff = 1` and `l_linger = 0`: aborts connection
③ `l_onoff = 1` and `l_linger ≠ 0`: linger on `close`

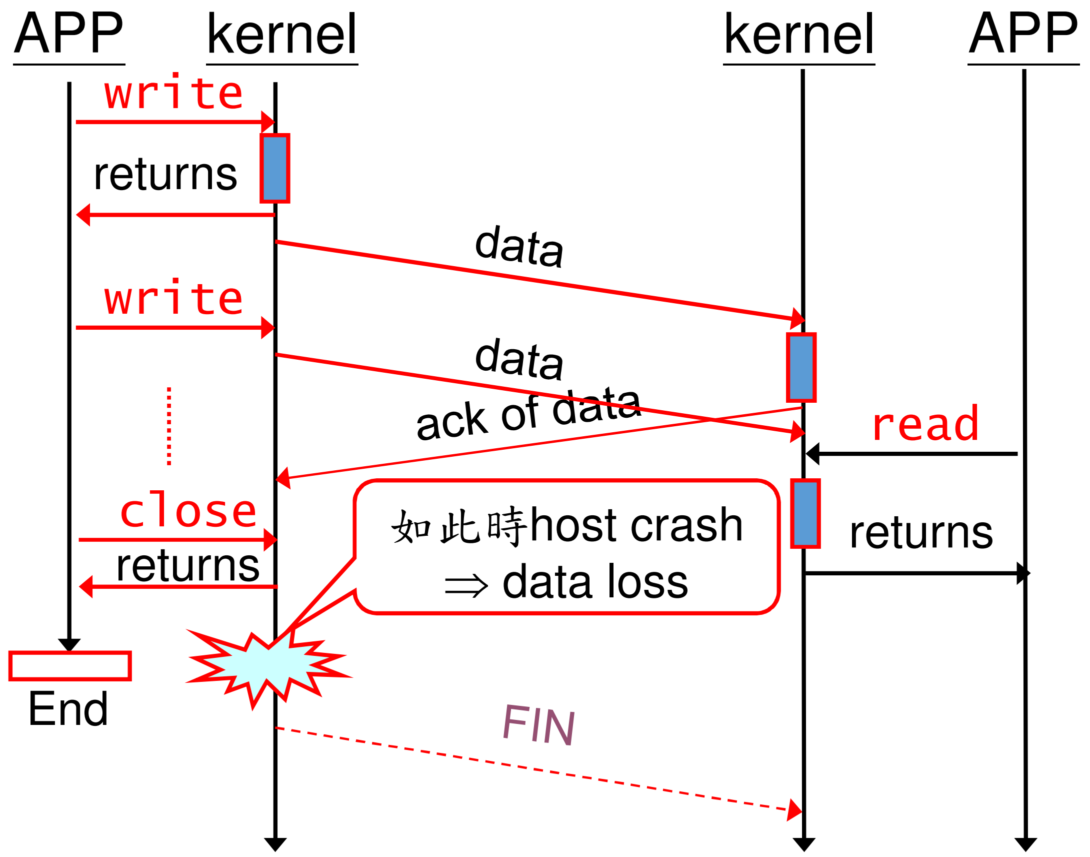
Option 1: `l_onoff = 0` (1/2)



Option 1: `l_onoff = 0` (2/2)



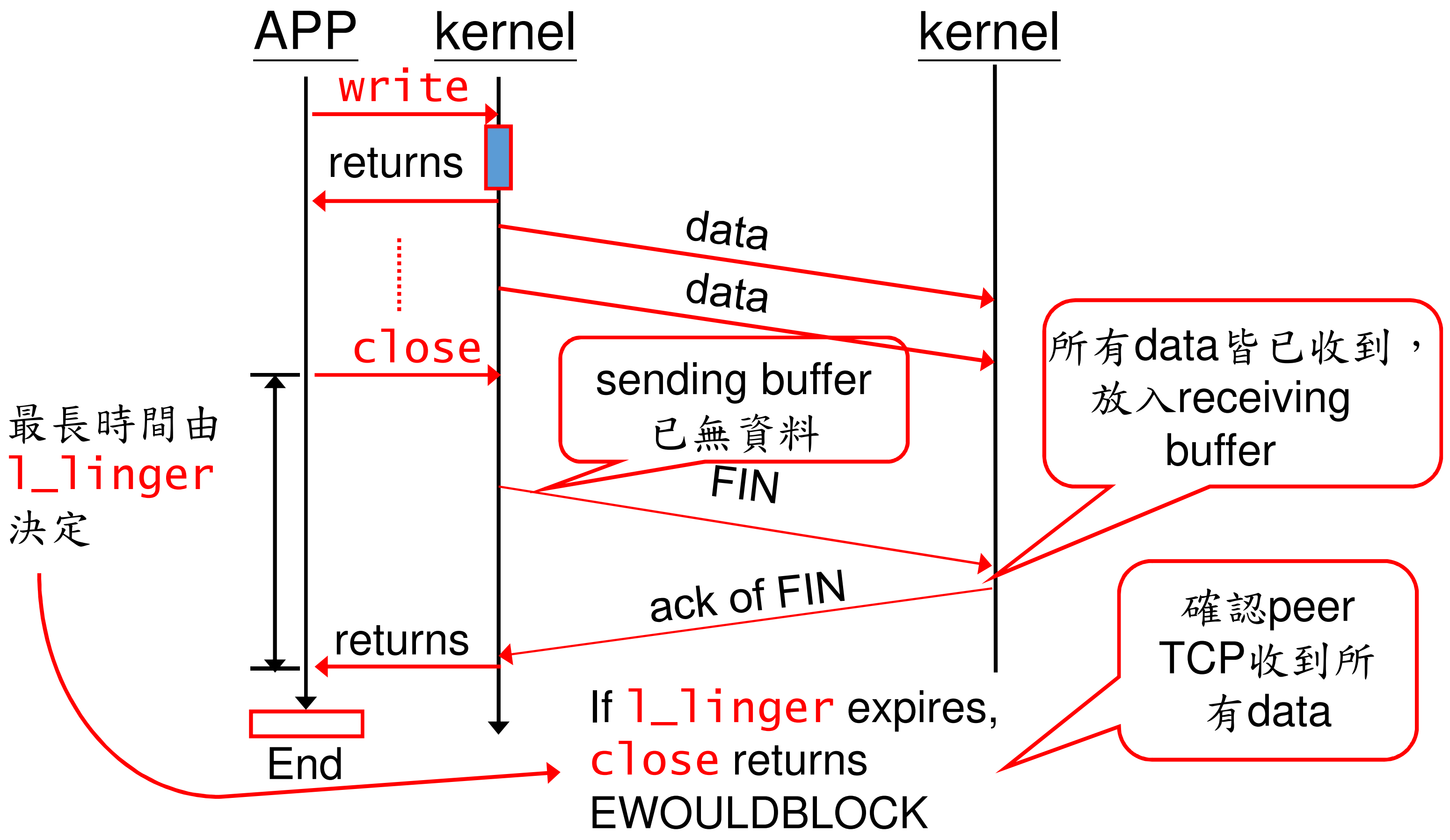
One Risk of `l_onoff = 0`



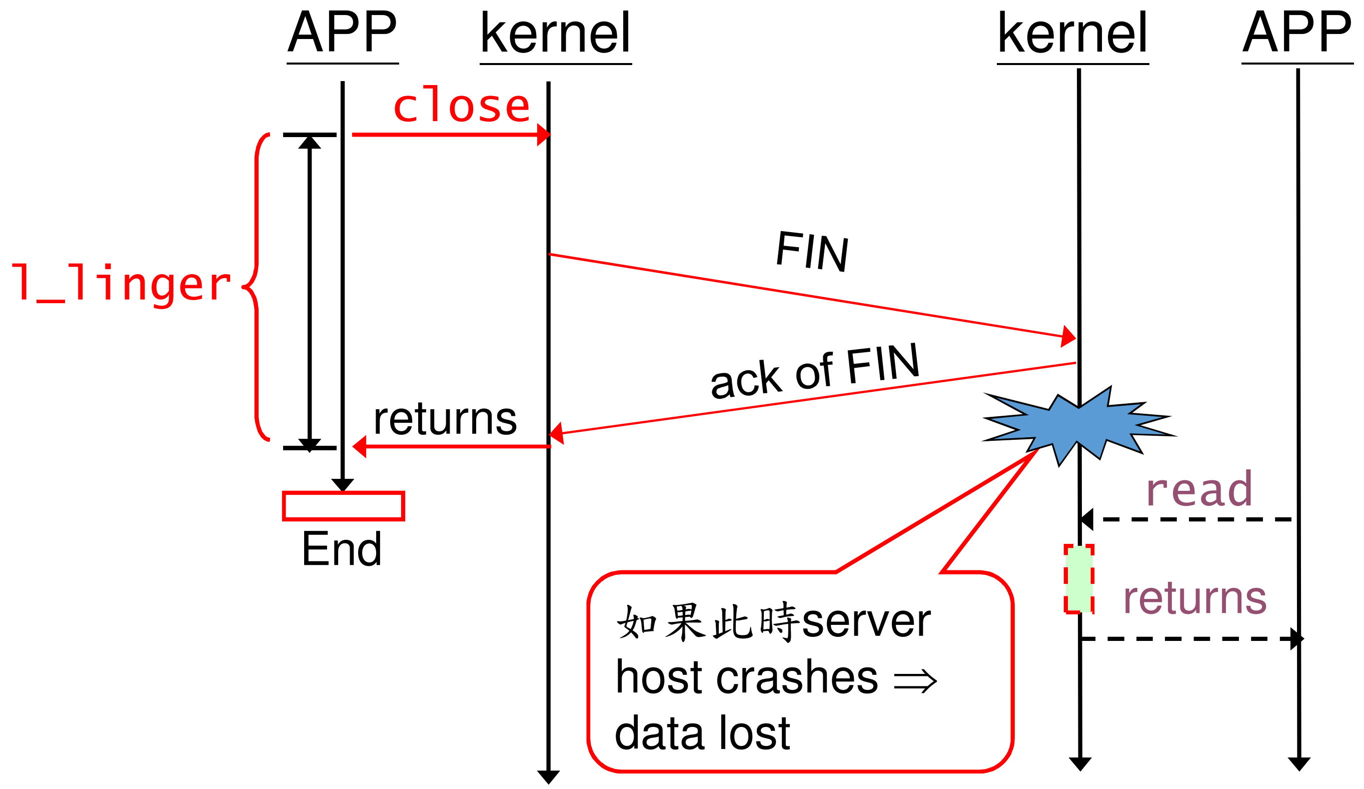
Option 2: `l_onoff = 1` And `l_linger = 0`

- TCP discards any data still remaining in the socket send buffer (這些data對應的ack都尚未收到), and sends an RST to the peer TCP
- This avoids TCP's TIME_WAIT state, which may cause problems.

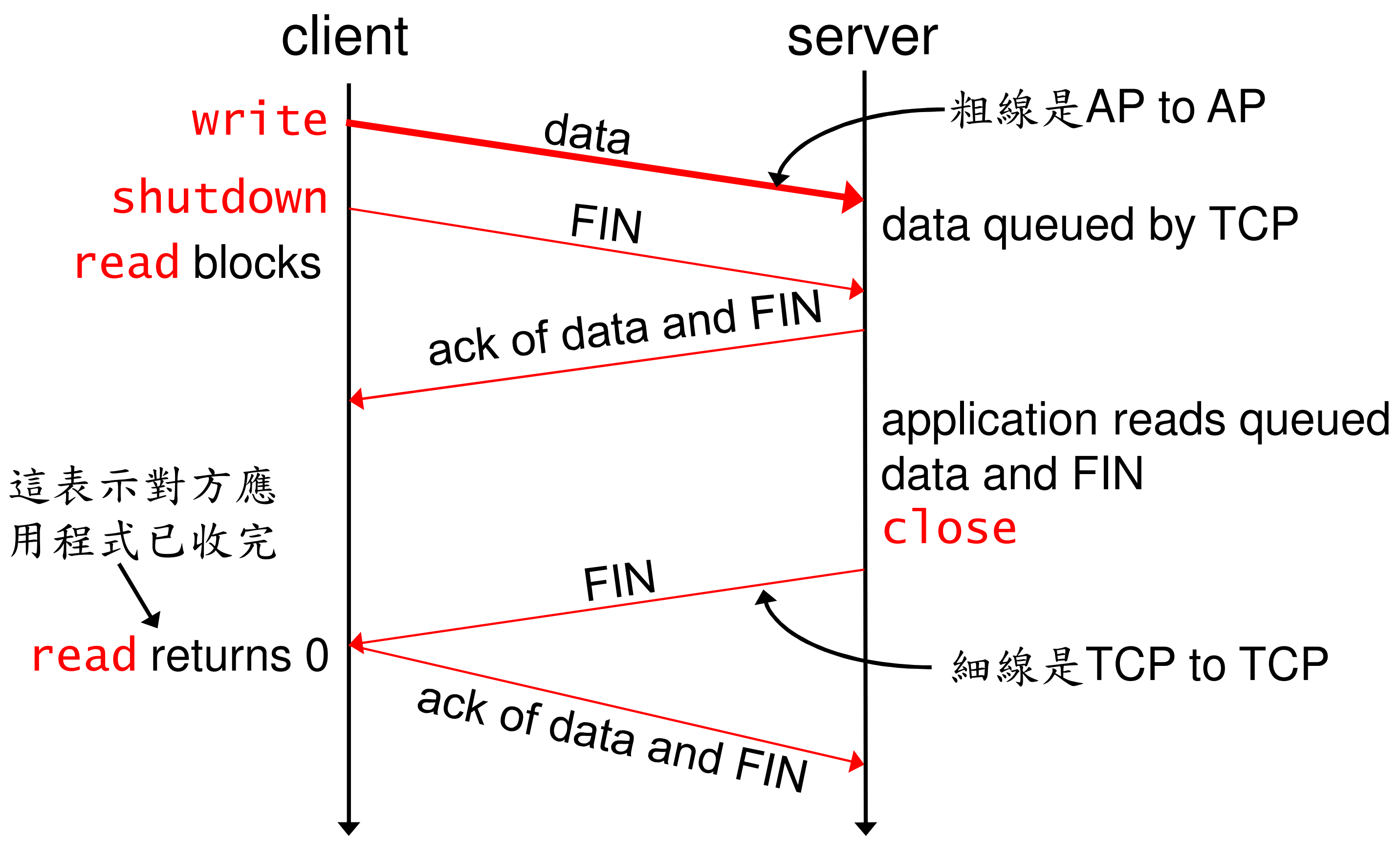
Option 3: `l_onoff = 1` And `l_linger ≠ 0`



Risk of Option 3: `l_onoff = 1` and `l_linger ≠ 0`



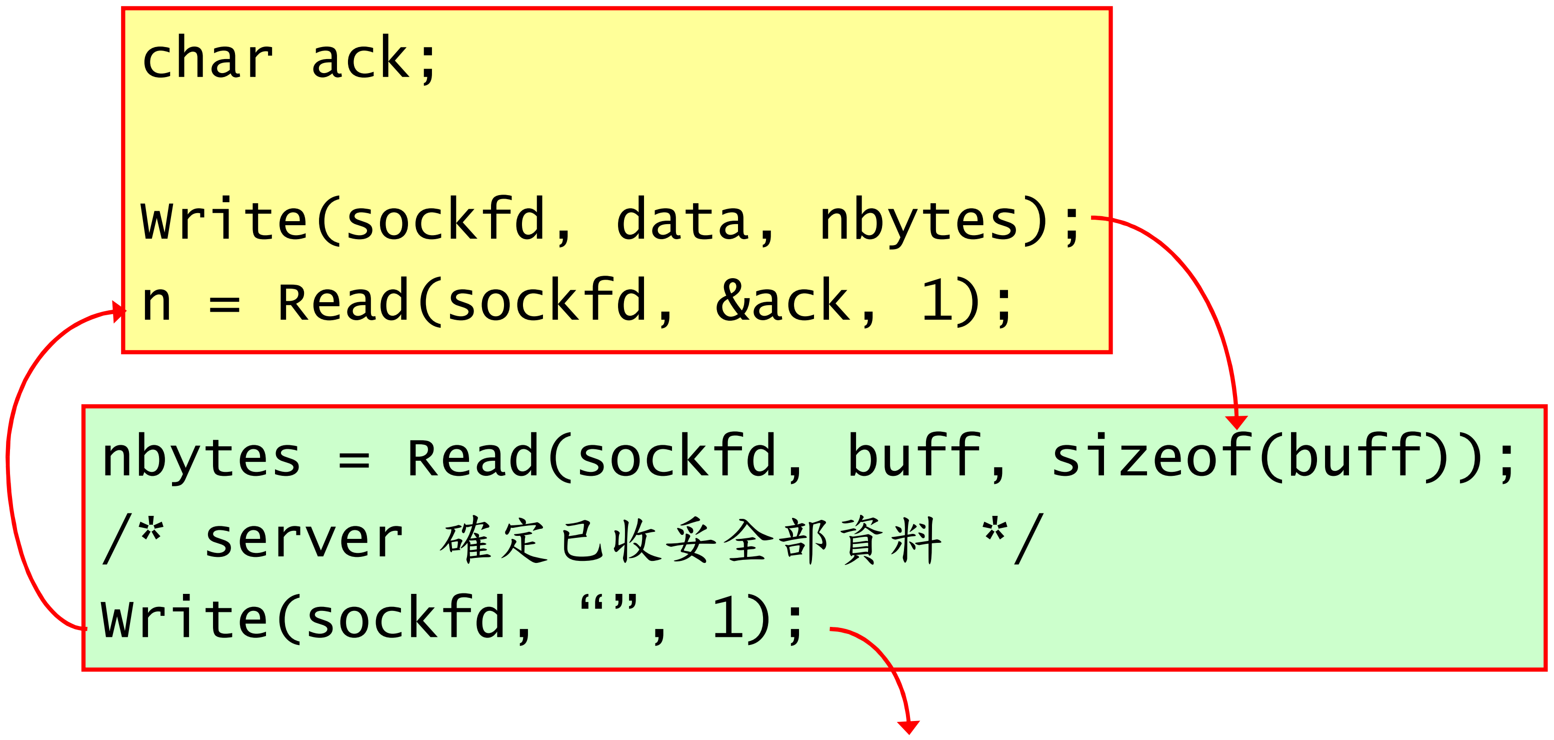
To Ensure That Peer *Process* Has Read Our Data: Using `shutdown`



When We Close Our End of the Connection…

- The return can occur at three different times
  - `close` returns immediately, without waiting at all (application knows nothing)
  - `close` lingers until the ACK of our FIN is received (peer TCP has received all data)
  - `shutdown` followed by a `read` waits until we receive the peer's FIN (peer process has received all data)

Another Alternative: Using Application-Level Acknowledgement



```
char ack;

Write(sockfd, data, nbytes);
n = Read(sockfd, &ack, 1);
```

```
nbytes = Read(sockfd, buff, sizeof(buff));
/* server 確定已收妥全部資料 */
Write(sockfd, “”, 1);
```

Generic Socket Options (Cont.)

- SO_OOBINLINE: leave received out-of-band data inline in the normal input queue
- SO_RCVBUF/SO_SNDBUF: socket receive / send buffer size, TCP default: 8192-61440, UDP default: 40000/9000
- SO_RCVLOWAT/ SO_SNDLOWAT: receive / send buffer low water mark for *select* to return

Generic Socket Options (Cont.)

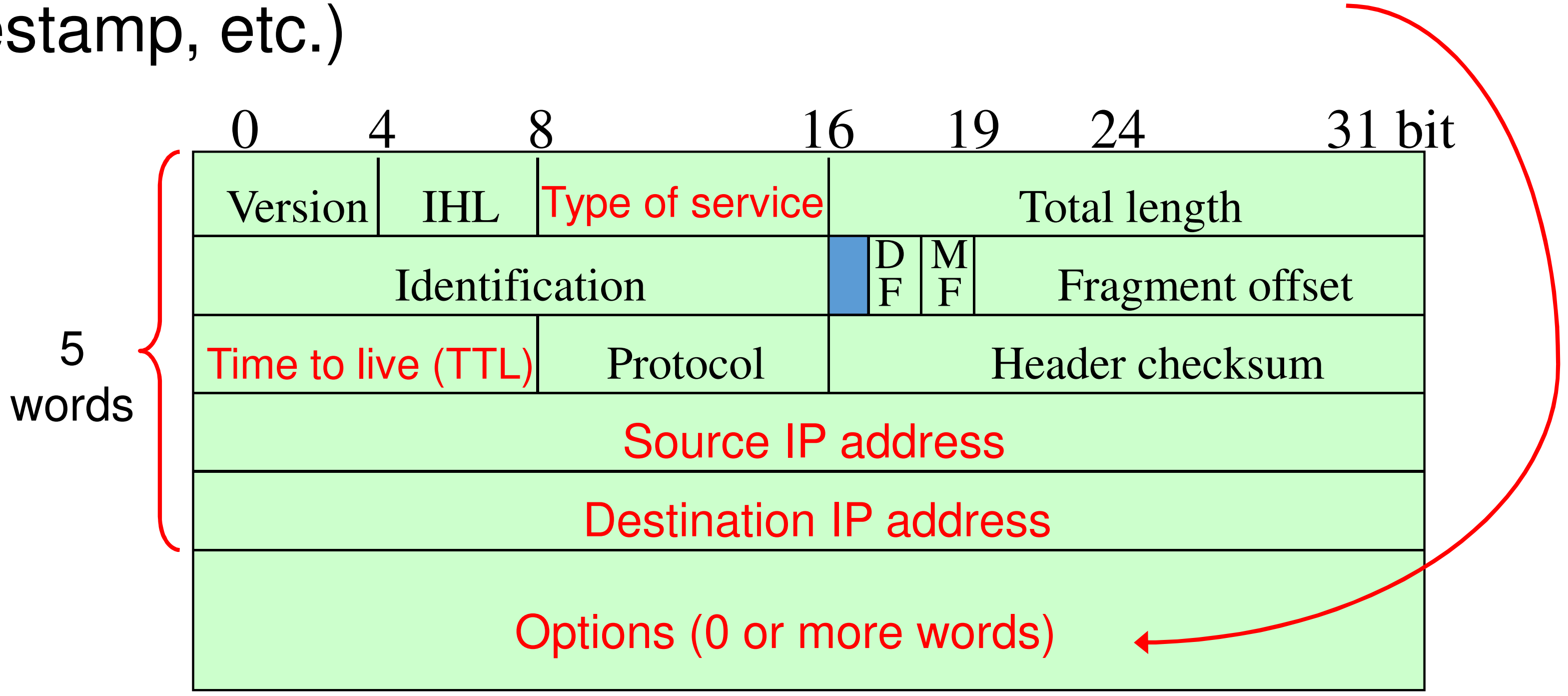
- SO_RCVTIMEO/SO_SNDTIMEO: receive / send timeout for socket read/write functions
- SO_REUSEADDR/SO_REUSEPORT: allow local address reuse for TCP server restart, IP alias, UDP duplicate binding for multicasting
- SO_TYPE: get socket type (SOCK_STREAM or SOCK_DGRAM)
- SO_USELOOPBACK: applies only to routing socket; gets copy of what it sends (not a POSIX standard)

IPv4 Socket Options

- IP_HDRINCL: If set for a raw IP socket, we must build our own IP header for all the datagrams that we send on the raw socket.
  - Normally the kernel builds the IP header
- e.g. *traceroute* builds its own IP header on a raw socket

IPv4 Socket Options (Cont.)

- IP_OPTIONS: specify IP options in the IPv4 header (source route, timestamp, etc.)



IPv4 Socket Options (Cont.)

- IP_RECVSTADDR: return destination IP address of a received UDP datagram by `recvmsg` (Ch. 8)
- IP_RECVIF: return received interface index for a received UDP datagram by `recvmsg`
- IP_TOS: set IP Type of Service (TOS) field of outgoing packets for TCP/UDP socket.
  - options: IPTOS_LOWDELAY, IPTOS_THROUGHPUT, IPTOS_RELIABILITY, IPTOS_LOWCOST

IP Socket Options (Cont.)

- IP_TTL: set and fetch the default TTL for outgoing packets, 64 for TCP/UDP sockets, 255 for raw sockets, used in *traceroute*
- IP_MULTICAST_IF, IP_MULTICAST_TTL, IP_MULTICAST_LOOP,IP_ADD_MEMBERSHIPIPIIP_DROP_MEMBERSHIP (Sec. 19.5)

IPv6 Socket Options

- ICMP6_FILTER: fetch and set `icmp6_filter` structure specifying message types to pass
- IPV6_ADDFORM: change address format of socket between IPv4 and IPv6
- IPV6_CHECKSUM: offset of checksum field for raw socket
- IPV6_DSTOPTS: return destination options of received datagram by `recvmsg`
- IPV6_HOPLIMIT: return hop limit of received datagrams by `recvmsg`

IPv6 Socket Options (Cont.)

- IPV6_HOPOPS: return hop-by-hop options of received datagrams by `recvmsg`
- IPV6_NEXTHOP: specify next hop address as a socket address structure for a datagram
- IPV6_PKTINFO: return packet info, dest IPv6 address and arriving interface, of received datagrams by `recvmsg`

IPv6 Socket Options (Cont.)

- IPV6_PKTOPTIONS: specify socket options of TCP socket (UDP uses recvmsg and sendmsg)
- IPV6_RTHDR: receive routing header (source route)
- IPV6_UNICAST_HOPS: ~ IP_TTL
- IPV6_MULTICAST_IF/HOPS/LOOP, IPV6_ADD/DROP_MEMBERSHIP (Sec. 19.5)

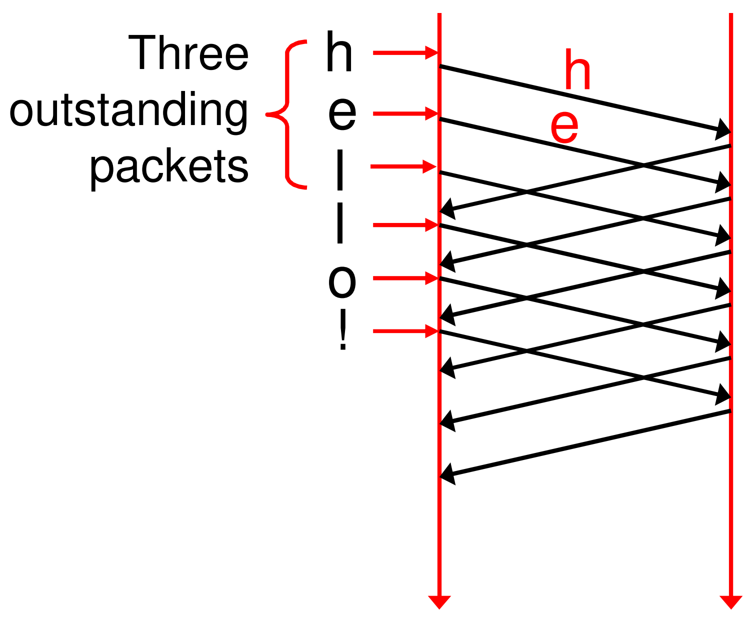
TCP Socket Options

- TCP_KEEPALIVE: seconds between probes
- TCP_MAXRT: TCP max retx time
- TCP_MAXSEG: TCP max segment size
- TCP_NODELAY: disable Nagle algorithm (Nagle algorithm reduces the number of small packets)
- TCP_STDURG: interpretation of TCP's urgent pointer, used with out-of-band data

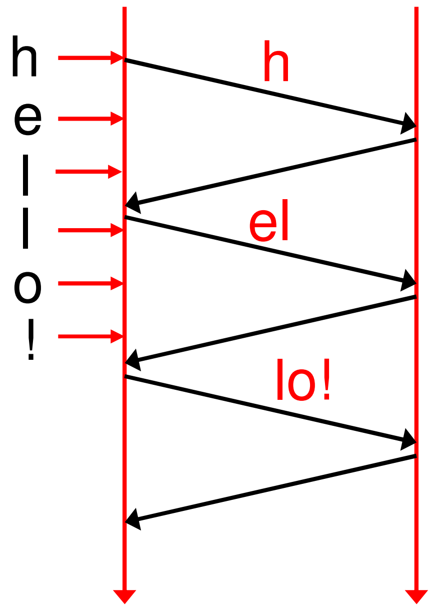
Nagle Algorithm

- Reduce the number of small packets on a WAN
- outstanding data:已送出但尚未收到ACK的data
- 如果某連線已有outstanding data，則應用程式要求TCP送出的small data不會被送出，直到outstanding data的ACK收到
- 避免連線同時存在多個small outstanding data

Enable/Disable Nagle Algorithm的差異



disabled



enabled
(default)

When should we disable Nagle Algo.?

`fcntl` Function

- Calling `fcntl` is a preferred way to
  - Set socket for non-blocking I/O (第二種I/O)
  - Set socket for signal-driven I/O (第四種I/O)
  - Set socket owner
    - Socket owner 是接收 SIGIO 和 SIGURG 信號的 process
  - Get socket owner

fcntl Function Prototyping

```
#include <fcntl.h>
int fcntl (int fd, int cmd, ... /* int arg */ );
            returns: depends on cmd if OK,
                                 -1 on error
```

cmd: F_GETFL: get flag
  F_SETFL: set flag

Two flags that affect a socket
- O_NONBLOCK (nonblocking I/O) 第15章
- O_ASYNC (signal-driven I/O notification)

To Set a Flag

To set a flag : ①fetch ②OR/AND~ ③set

```
int flags;
if ( (flags = fcntl(fd, F_GETFL, 0) ) < 0)
    error_sys ("F_GETFL error");
flags |= O_NONBLOCK; /* turn on */
( or flags &= ~O_NONBLOCK; /* turn off */ )
if (fcntl(fd, F_SETFL, flags) < 0)
    error_sys ("F_SETFL error");
```

Set and Get Socket Owner

- F_SETOWN command lets us set socket owner (process ID or process group ID) to receive SIGIO and SIGURG signals
  - **SIGIO** is generated if signal-driven I/O is enabled for a socket
  - **SIGURG** is generated when a out-of-band data arrives for a socket.
- F_GETOWN command gets socket owner

Summary

- Commonly used socket options: `SO_KEEPALIVE`, `SO_RCVBUF`, `SO_SNDBUF`, `SO_REUSEADDR`
- `SO_REUSEADDR` set in TCP server before calling `bind`
- `SO_LINGER` gives more control over when close returns and forces RST to be sent
- `SO_RCVBUF`/`SO_SNDBUF` for bulk data transfer across long fat pipes