Elementary Socket Options

Prof. Anand D Dave
Department of Information Technology,
Dharmsinh Desai University, Nadiad.

Elementary Socket Options

- They are used to enable/disable features to get more control on behavior of sockets.
- They are used to change default value of some feature
- Two functions
  - getsockopt
  - setsockopt

Elementary Socket Options

- getsockopt and setsockopt function

```
#include <sys/socket.h>
int getsockopt(int sockfd, int level, int optname, void *optval,
socklent_t *optlen);

int setsockopt(int sockfd, int level , int optname, const void *optval,
    socklent_t optlen);
```

- sockfd: open socket descriptor
- level: code in the system to interpret the option(generic, IPv4, IPv6, TCP)
  - level includes general socket option (SOL_SOCKET) and protocolspecific option (IP, TCP, etc)

Elementary Socket Options

- getsockopt and setsockopt function
- optval: pointer to a variable from which the new value of option is fetched by setsockopt, or into which the current value of the option is stored by getsockopt.
  - Option value can be of different types : int, in_addr, timeval, …
  - that is the reason we use the void pointer.
- optlen: the size of the option variable

Elementary Socket Options

- getsockopt and setsockopt function

  - Two types of options

  – Binary option

  - Used to enable or disable certain feature (flag option).
  - optval is integer type.
  – In getsockopt, The returned optval 0 means option is disabled and 1 means it is enabled.
  – In setsockopt, optval 0 is used to disable the option and nonzero is used to enable the option.
  – Value option

  - Uses specific values.
  - Used to pass/fetch values, structures, etc

Elementary Socket Options

Socket options

General socket options (level = SOL_SOCKET)

- Optnames:

– SO_BROADCAST(int): permit sending broadcast datagram
– SO_ERROR: can only be got (not set), reset the error
– SO_KEEPALIVE: for TCP only, automatic send keepalive message when inactive for 2 hours (can be modified).
– SO_LINGER: for TCP only, determines the behavior when a close is called.
– SO_RCVBUF, SO_SNDBUF: send and receive buffer size.

Elementary Socket Options

Socket options

IP options:
– Allows packets sent through a socket to have certain behavior,
– Level = IPPROTO_IP
– E.g., manipulating IP header fields

Elementary Socket Options

Socket options

TCP options:

- Level = IPPROTO_TCP
- Optnames:
  - TCP_KEEPALIVE: set the time
  - TCP_NODELAY: wait to ack or not (enable)

Elementary Socket Options

Inheritance of Socket options

The following socket options are inherited by a connected TCP socket from the listening socket.

- SO_KEEPALIVE
- SO_LINGER
- SO_RCVBUF
- SO_SNDBUF

• When to set these options?

- The connected socket is returned to a server by accept until the three-way handshake is completed by the TCP layer.
- Therefore if something should happen before 3WH completes ?
- To set these option(s), set them for the listening socket.

Elementary Socket Options

Generic Socket Options

- These options are protocol independent. However, certain options apply to only certain types of sockets (E.g.,SO_BROADCAST).
- They are handled by a protocol independent code within the kernel. (Not by one particular protocol module)

Elementary Socket Options

Generic Socket Options (SO_BROADCAST)

- It enables or disables the ability of the process to send broadcast message on broadcast links.
  - Can work only with datagram socket.
  - Moreover only on network that supports the concept of broadcast.
    - E.g Ethernet, token ring..)
- Example:

```
int broadcastFlag=1;
setsockopt(sd,SOL_SOCKET,SO_BROADCAST,&broadcastFlag,size of(broadcastFlag));
```

- If the destination address is a broadcast address, and this socket option is not set, EACCES is returned.

Elementary Socket Options

Generic Socket Options (SO_ERROR)

- when error occurs on a socket, the protocol module in a Berkeley-derived kernel sets one of standard UNIX Exxx values to a variable named so_error for that socket. Not errno, it is error of a process.
  - It is called *pending error* for the socket.
  - The process can be immediately notified of the error in one of two ways
  - If the process is blocked in a call to select on the socket
  - If the process is using signal-driven I/O, the SIGIO signal is generated for either the process or the process group.
  - The process can then obtain the value of so_error by fetching the SO_ERROR socket option.

Elementary Socket Options

Generic Socket Options (SO_ERROR)

- If so_error is nonzero when the process calls read And there is no data to return, read returns -1 with errno set to the value of so_error. The value of so_error is then reset to 0.
  - If there is data queued for the socket, that data is returned by read instead of error condition.
- If so_error is nonzero when the process calls write, -1 is returned with errno set to the value of so_error. The so_error is reset to 0.

Elementary Socket Options

Generic Socket Options (SO_KEEPALIVE)

- It is used to detect peer host crash.
- When the keepalive option is set for a TCP socket.
  - And there is no data exchange for 2hours,
  - then TCP automatically sends a keep-alive probe to the peer.
- Possible peer responses of keep-alive probe
  - ACK(everything OK)
  - RST(peer crashed and rebooted):ECONNRESET
  - no response to keep alive probe.

Elementary Socket Options

Generic Socket Options (SO_KEEPALIVE)

- Peer response-1: ACK
  - The peer TCP responds with the expected ACK.
  - Everything OK, and application is not notified.
  - TCP will send another probe following another 2 hours of inactivity.

Elementary Socket Options

Generic Socket Options (SO_KEEPALIVE)

- Peer response-2: RST
  - The peer TCP responds with RST.
  - It indicates that peer host has crashed and rebooted.
  - Socket's pending error is set to ECONNRESET.
    - When do we get this error? See SO_ERROR.
  - And the socket is closed.

Elementary Socket Options

Generic Socket Options (SO_KEEPALIVE)

Peer response-3: No response to keep alive probe.

– Berkely derived TCPs send eight additional probes, 75 seconds apart, trying to elicit response.

– TCP will give up if no response within 11 minutes and 15 seconds after sending the first probe.

– If there is no response at all to TCP's keepalive probes,

• Socket's pending error is set to ETIMEDOUT and the socket is closed. Or

• If the socket receives an ICMP error in response to one of keepalive probes, the corresponding error is returned instead. (EHOSTUNREACH)

Elementary Socket Options

Generic Socket Options (SO_KEEPALIVE)

- Can we change inactivity duration (i.e., rather than 2 hours, can we specify some other period)?
  - Most kernels maintain these parameters on a per-kernel basis.
  - So, changing the inactivity period from 2 hours to 15 minutes will affect all sockets on the host.

Elementary Socket Options

Generic Socket Options (SO_KEEPALIVE)

Use of SO_KEEPALIVE to detect peer host crash.

- If the peer process crashes?

– Its TCP will send a FIN across the connection, which we can easily detect with select (through I/O multiplexing).

- If there is no response to any keep-alive probes?

– We are not guaranteed that peer host has crashed.

– It could be possible that some intermediate router failed for 15 minutes (for example,), and our probe sending period gets completely overlapped by this 15 minutes period.

Elementary Socket Options

Generic Socket Options (SO_KEEPALIVE)

- Practical usage
  - This option is normally used by servers.
  - Servers use the option as they spend most of their time for waiting for input across the TCP connection.
  - If the client host crashes, the server process will never know about it.
  - And the server will continually wait for input that can never arrive.
  - This is called half-open connection. The keep-alive option will detect these half-open connections and terminate them.

Elementary Socket Options

Generic Socket Options (SO_LINGER)

Linger means gradually dying.

- Using this option, we can specify how the close function operates for a connection oriented protocol.
- By default, close returns immediately, but if there is any data still remaining in the socket send buffer, the system will try to deliver the data to the peer.
- We can change this default behavior by passing properly initialized struct linger structure to setsockopt function.

Elementary Socket Options

Generic Socket Options (SO_LINGER)

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

- l_onoff = 0 : turn off the option , l_linger is ignored.
  - We get default behavior.
- l_onoff = nonzero and l_linger is 0:
  - TCP aborts the connection when the close is called.
  - TCP discard any remaining data in the socket send buffer and sends RST to the peer, not the normal four packet connection termination sequence.
  - Moreover, TCP's TIME_WAIT state is avoided. There is possibility of another incarnation of this connection gets related within 2MSL seconds (old duplicates from earlier connection arrive at new connection).

Elementary Socket Options

Generic Socket Options (SO_LINGER)

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

- l_onoff = nonzero and l_linger is nonzero :
  - Kernel will linger until socket is closed.
  - If there is any data still remaining in socket send buffer, the process is put to sleep until
    - Either all the data is sent and acknowledge by peer TCP
    - or until the linger time expires.
  - If socket has been set nonblocking, it will not wait for the close to complete, even if linger time is nonzero.

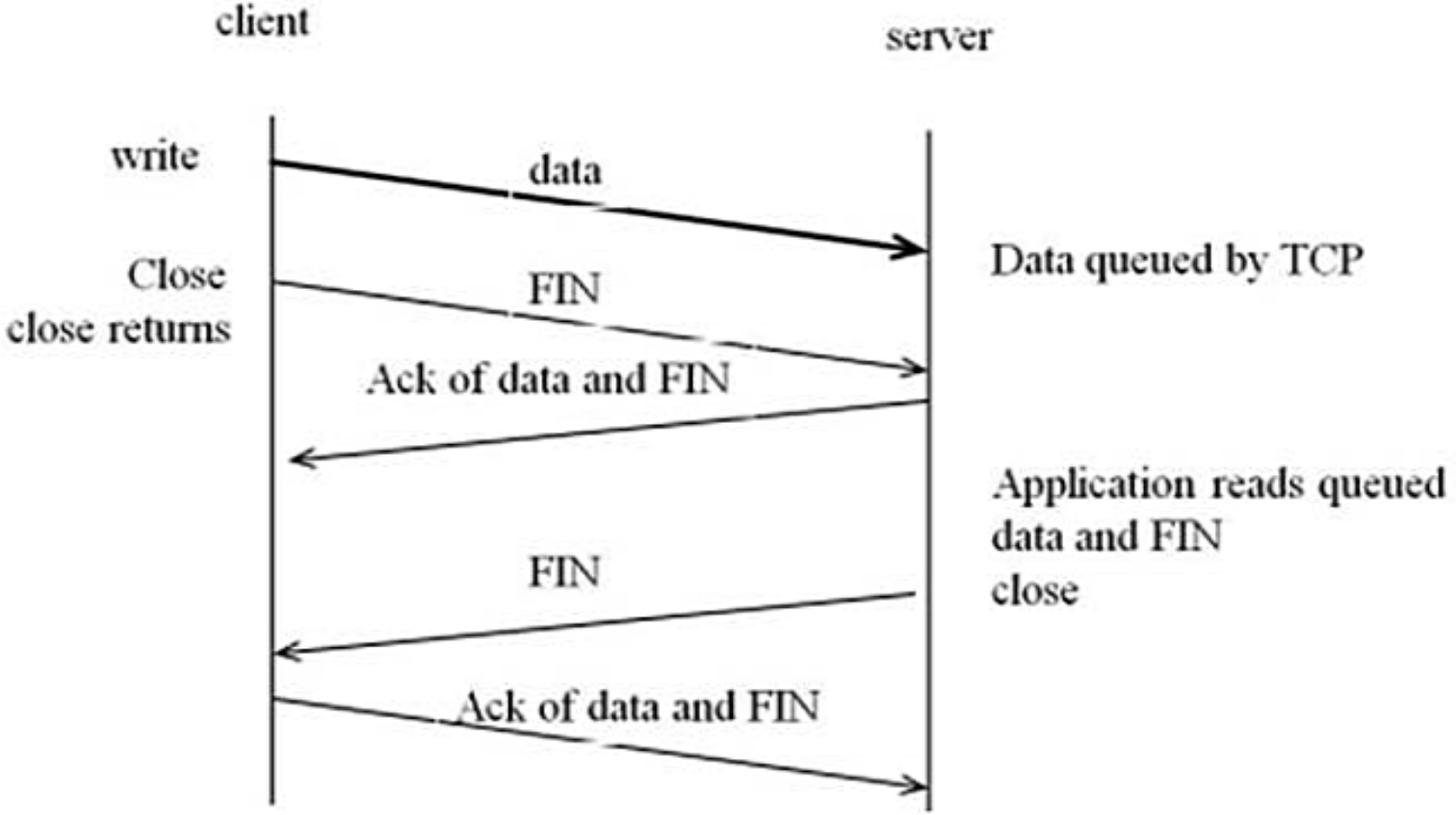
Elementary Socket Options

Generic Socket Options (SO_LINGER)

- The application should check the return value from close while using linger socket option.
- If the linger time expires before the remaining data is sent and acknowledged, close returns EWOULDBLOCK and any remaining data in the send buffer is discarded.

Elementary Socket Options

- Default operation of close. It returns immediately

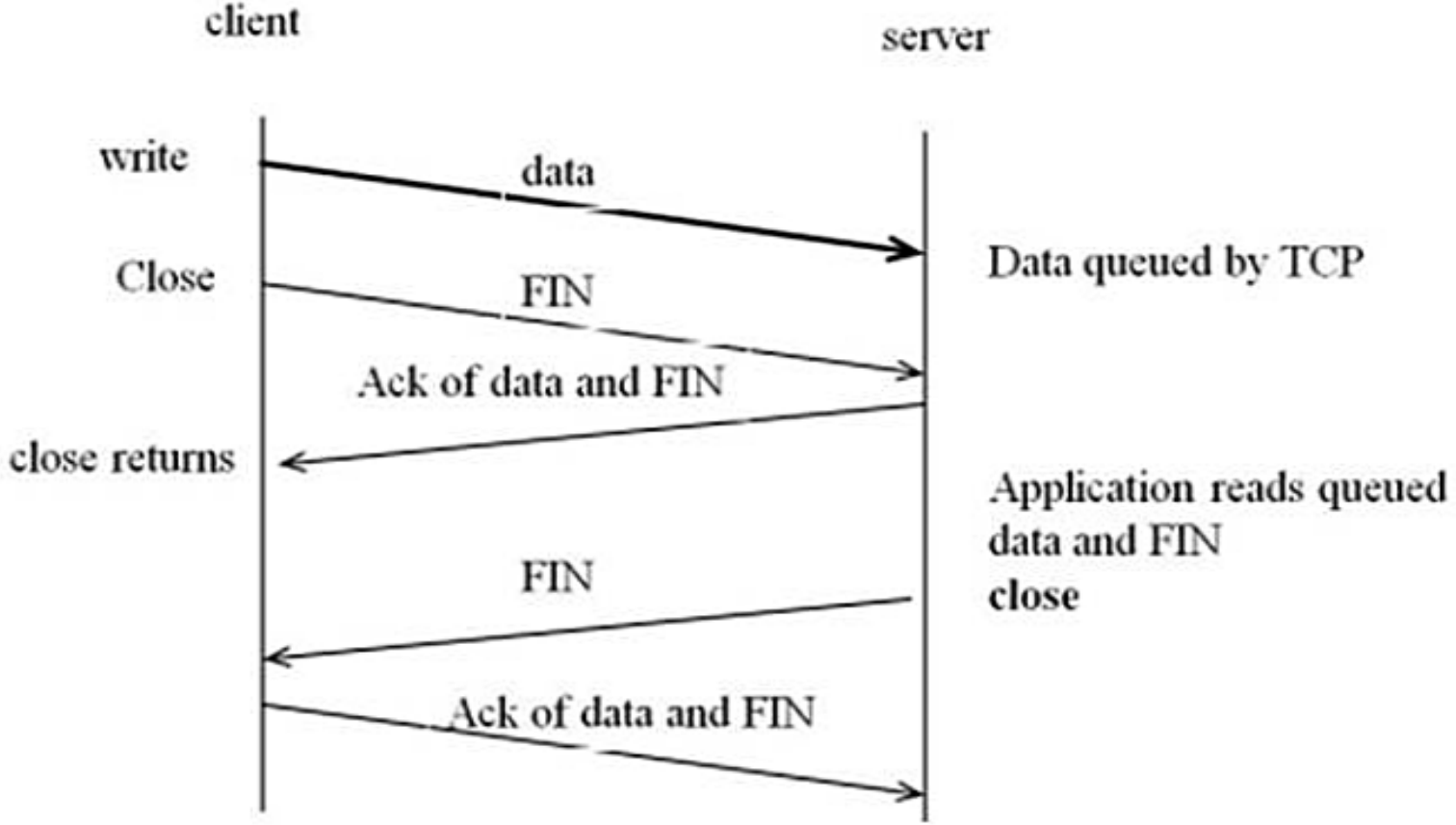


Elementary Socket Options

- In default behavior, there exists two problems.
  - It is possible that the client's close can return before the server reads the remaining data in its socket receive buffer.
  - It is possible that the server host crashes before server application reads this remaining data, and the client application will never know this.

Elementary Socket Options

- By setting so_linger socket option, the client can make sure that its sent data and FIN have been acknowledged by the peer TCP. But still, the problem 2 exists.



Elementary Socket Options

- Successful return from close, with SO_LINGER option set, only tells us that the data we sent and our FIN have been acknowledged by the peer TCP.
  - This does not tell us whether the peer application has read the data.

Elementary Socket Options

- How to make sure that the peer application has read our data?
  - Use of shutdown
  - Use of application level acknowledgement of data.
  - shutdown

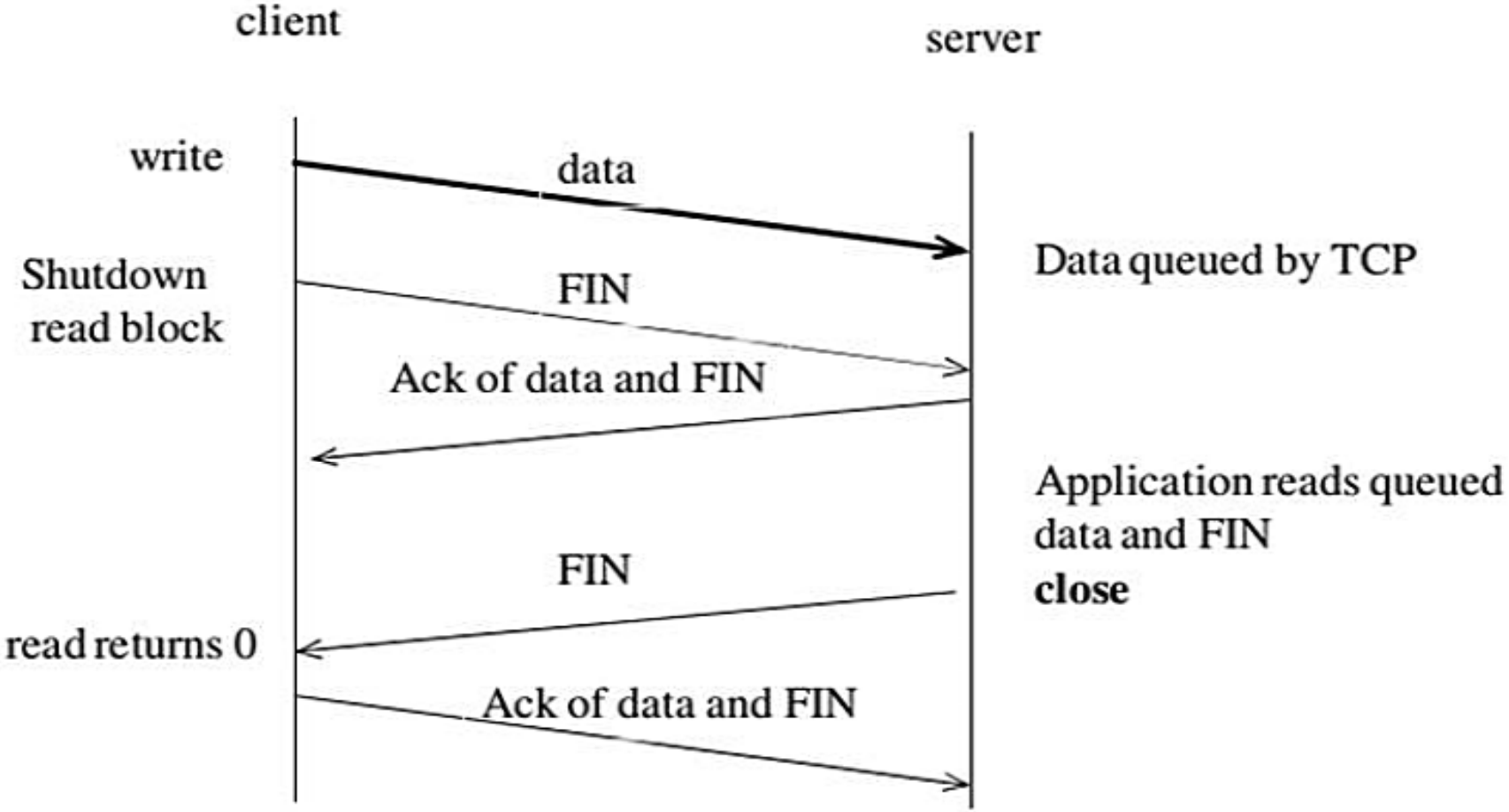
```
#include <sys/socket.h>
int shutdown(int s, int how);
```

(On success, zero is returned. On error, -1 is returned, and errno is set appropriately.)

  - If how = SHUT_RD, further receiving will not be allowed.
  - If how = SHUT_WR, further sending will not be allowed.
  - If how = SHUT_RDWR, further receiving and sending will not be allowed.

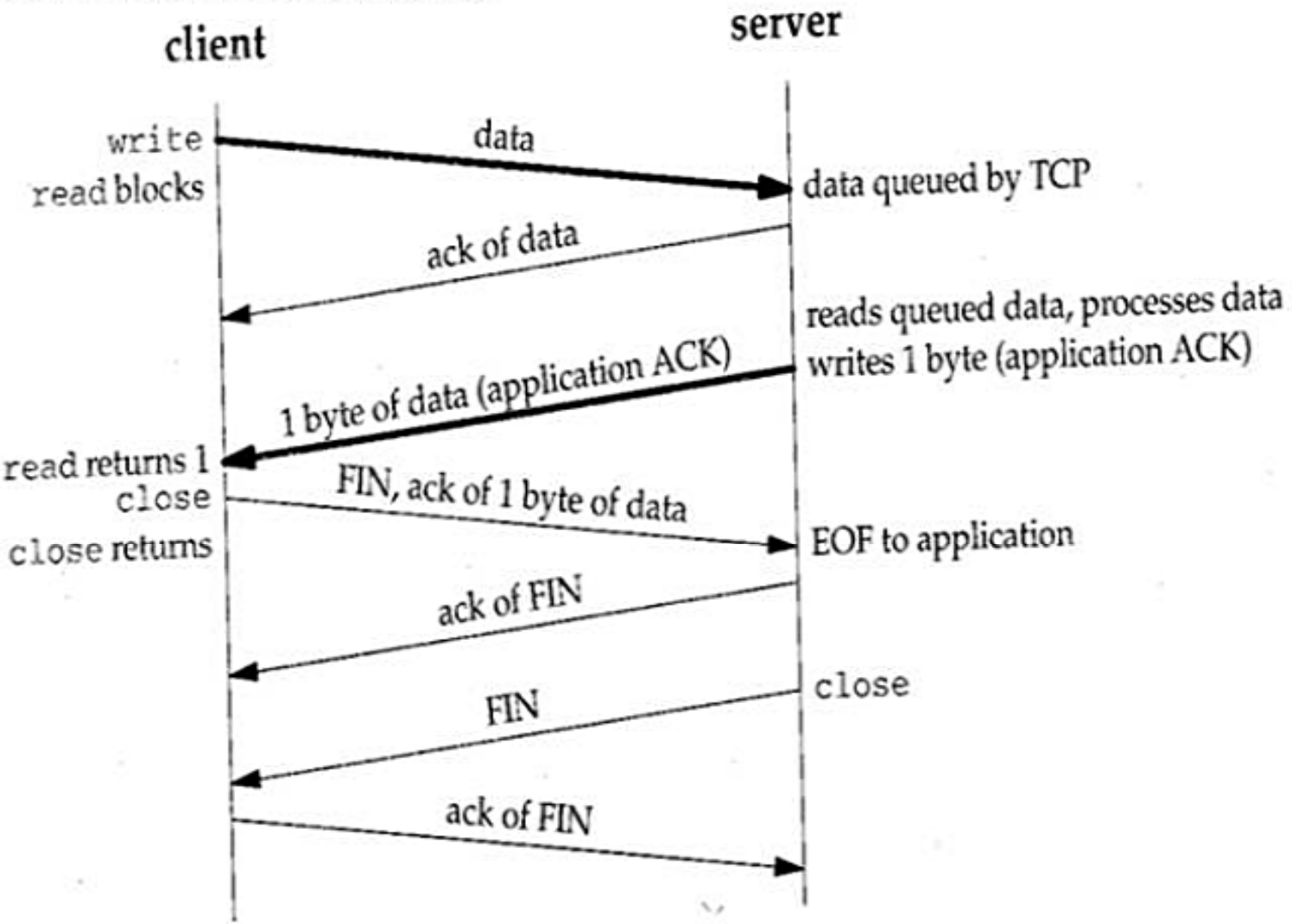
Elementary Socket Options

- **Use of shutdown** to know the peer application has read our data. Call shutdown with how=SHUT_WR instead of close and then wait for the peer to close its end of the connection.



Elementary Socket Options

- **Use of application level acknowledgement of data** to know the peer application has read our data.



Elementary Socket Options

- **Use of application level acknowledgement of data** to know the peer application has read our data.

  - Use ack of 1 byte.
  - Client code

```
char ack;
write(sockfd, data, nbytes); /* data from client to server */
N=read(sockfd, &ack, 1); /* wait for application-level ack */
```

  - Server

```
nbytes=read(sockfd, buff, sizeof(buff)); /*data from client */
/* server verifies it received the correct amount of data from the client */
write(sockfd, " ", 1); /* server's ACK back to client */
```

References

- Sources :
- https://www.javatpoint.com/
- https://www.tutorialspoint.com/
- https://sites.google.com/site/prajapatiharshadb/class-notes-for-students
- UNIX Network Programming by W. Richard Stevens, Prentice Hall Publication
- Distributed Computing: Concepts &Applications: by M. L. Liu Addison Wiselly