Sockets 3: I/O Syscalls

1. Introduction and Addresses
2. Basic System Calls
3. **Socket I/O Syscalls**
   - **write, send, sendto, sendmsg**
   - **read, recv, recvfrom, recvmsg**
4. IP Addresses and Hostnames
5. Endianess, Options, Servers

write, send, sendto, sendmsg

`write()` can be used with *connected sockets*.

There are other calls that provide additional functionality or must be used with *unconnected sockets*:

- `send` – similar to `write()`
- `sendto` – includes target address, for unconnected sockets
- `sendmsg` – multiple buffers, etc.

TCP (`SOCK_STREAM`) sockets are necessarily connected.

UDP (`SOCK_DGREAM`) sockets are unconnected by default, but can become connected by using `connect()`.

send

Like `write()`, `send()` must be used with *connected sockets*.

Identical in function to `write()` with the addition of the `flags` (options) parameter:

```
ssize_t send(int s, const void *buf, size_t len, int flags)
```

Among the values for use in `flags` are:

- `MSG_DONTWAIT` – make operation **non-blocking**
- `MSG_OOB` – send **out-of-band** data
- `MSG_EOR` – terminates a record, for sockets of type `SOCK_SEQPACKET`
- `MSG_NOSIGNAL` – do *not* generate `SIGPIPE` signal on errors

sendto

`sendto()` is the primary call used with *unconnected sockets* (e.g., UDP).

It includes the address to send the message to:

```
ssize_t sendto(int s, const void *buf, size_t len, int flags,
               const struct sockaddr *to, socklen_t tolen)
```

read, recv, recvfrom, recvmsg

`read()` can be used with *connected* sockets.

There are other calls that provide additional functionality or must be used with *unconnected sockets*:

- `recv` – similar to `read()`
- `recvto` – includes source address, for unconnected sockets
- `recvmsg` – multiple buffers, etc.

recv

Like `read()`, `recv()` must be used with *connected sockets*.

Differs from `read()` only in the addition of the `flags` parameter:

```
ssize_t recv(int s, void *buf, size_t len, int flags)
```

Among the values for `flags` are:

- `MSG_DONTWAIT` – make operation **non-blocking**
- `MSG_OOB` – send **out-of-band** data
- `MSG_PEEK` – do not remove received data from kernel buffer
- `MSG_WAITALL` – block until *full* request can be satisfied

recvfrom

`recvfrom()` is the primary call for use with *unconnected sockets* (e.g., UDP).

It returns the address that sent the message:

```
ssize_t recvfrom(int s, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen)
```

write/send and Closed Connections

The behavior of `write()`/`send()` when the connection has been closed, requires some additional explanation.

By default, a `write()`/`send()` to a closed (stream) connection causes a `SIGPIPE` signal to be generated and sent to the process.

The *default disposition* for `SIGPIPE` is *termination*.

This means that attempting to a `write()`/`send()` to a closed connection will cause the calling process to immediately terminate.

(Exactly the same behavior results with a `write()` to a *pipe* when all the pipe's read ends have been closed.)

write/send and Closed Connections (contd.)

There are two ways to change the behavior of `write()`/`send()` so that they return the error `EPIPE` without being terminated by a `SIGPIPE` signal:

1. change the disposition of `SIGPIPE` to *ignored*:
   e.g., `signal(SIGPIPE, SIG_IGN)`
2. use the `MSG_NOSIGNAL` flag with each `send()` call:
   e.g., `send(sockfd, buff, bytes, MSG_NOSIGNAL)`

The difference is that `MSG_NOSIGNAL` applies *per-call*, while ignoring `SIGPIPE` sets a process attribute (so affects all `write()`/`send()` calls in *all threads* in the process).