Socket Layer Datasheet $Rev: 2$

A portable user-space implementation of the standard POSIX socket API, providing a modular backend interface for low-level network transport.

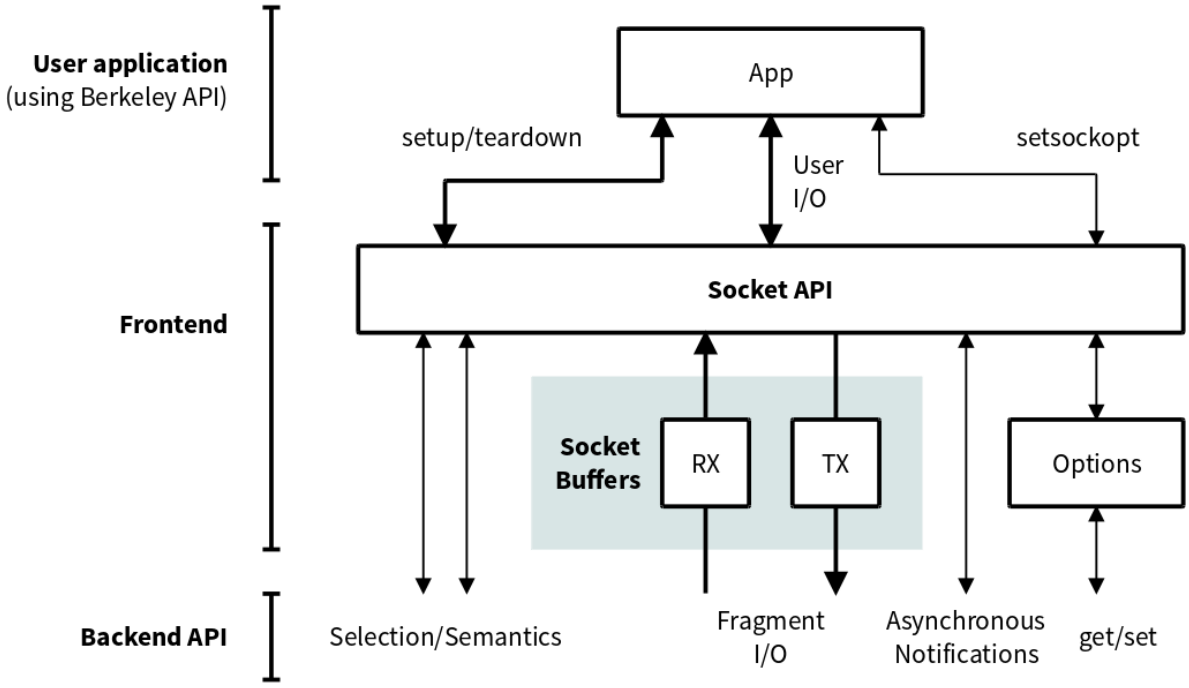
Description

This implementation of the standard POSIX socket API was motivated by the need to carry traffic over various physical transports. User traffic is fragmented and serialised before handing to the backend for transport. This allows for attaching the socket API to a TCP/UDP-layer engine or equivalent.

The frontend provides conformance to the standard API, and deals with maintenance behind sockets. Decisions on addressing are passed through to the backend, which can hook calls for fine-grained control.

Features

- Provides SOCK_STREAM & DGRAM
- Frontend is not specific to IP; this is not a TCP/UDP implementation.
- ISO C. Reentrant and thread-safe. Use existing bindings for other languages.
- User-space implementation, portable to various Operating Systems. Also suitable for non-hosted (embedded) systems.
- Can provide the socket API for systems which do not have it.
- Frontend API can be replaced with other socket APIs



Bubblephone Limited
Sussex Innovation Centre
Science Park Square
University of Sussex
Brighton BN1 9SB
Tel +44 (0) 1273 704 535

Backend API

All I/O formed into one read/write style interface; this is passed serialised fragments from the frontend. The backend implementation is responsible for supplying transport semantics, for example by TCP windowing or equivalent.

Asynchronous notifications are raised for incoming `accept` and `connect` requests. The backend API is separated into several parts:

Semantics

Notification of creation and destruction, with hooks for fine-grained control over various socket calls.

Selection

Exists for the lifetime of a socket, and responsible for addressing for that socket.

This maps addressing to a transport backend.

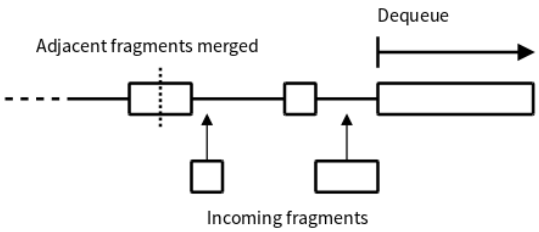
Fragmented I/O

Order, reliability and other transport behaviour must be provided by the back-end implementation for each socket type.

Socket Buffer

The Socket Buffer provides de-fragmentation and fragment ordering for SOCK_STREAM and SOCK_DGRAM. The implementation is a ring buffer, and may be operated at both ends independently. This allows for a multi-threaded application to use independent threads for the underlying network transport.

Streams

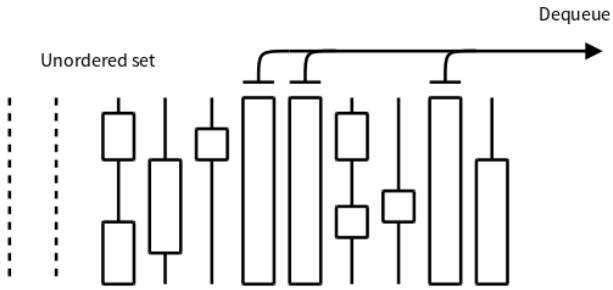


Streams have no limit to length; the start is open-ended.

Streams are a priority queue; fragments may arrive in no particular order. Adjacent fragments are merged when joining the queue.

The completed region dequeued up to the first hole.

Datagrams



Fragments within a datagram behave the same as fragments within a stream. In essence, each datagram is a fixed-length stream.

Datagrams may be different lengths (all illustrated here as the same length for simplicity).

Completed datagrams are dequeued in no particular order.