18. Interprocess Communication and Networking

The modules described in this chapter provide mechanisms for different processes to communicate.

Some modules only work for two processes that are on the same machine, e.g. `signal` and `mmap`. Other modules support networking protocols that two or more processes can use to communicate across machines.

The list of modules described in this chapter is:

- 18.1. `socket` — Low-level networking interface
  - 18.1.1. Socket families
  - 18.1.2. Module contents
    - 18.1.2.1. Exceptions
    - 18.1.2.2. Constants
    - 18.1.2.3. Functions
      - 18.1.2.3.1. Creating sockets
      - 18.1.2.3.2. Other functions
  - 18.1.3. Socket Objects
  - 18.1.4. Notes on socket timeouts
    - 18.1.4.1. Timeouts and the `connect` method
    - 18.1.4.2. Timeouts and the `accept` method
  - 18.1.5. Example
- 18.2. `ssl` — TLS/SSL wrapper for socket objects
  - 18.2.1. Functions, Constants, and Exceptions
    - 18.2.1.1. Socket creation
    - 18.2.1.2. Context creation
    - 18.2.1.3. Random generation
    - 18.2.1.4. Certificate handling
    - 18.2.1.5. Constants
  - 18.2.2. SSL Sockets
  - 18.2.3. SSL Contexts
  - 18.2.4. Certificates
    - 18.2.4.1. Certificate chains
    - 18.2.4.2. CA certificates
    - 18.2.4.3. Combined key and certificate
    - 18.2.4.4. Self-signed certificates
  - 18.2.5. Examples
    - 18.2.5.1. Testing for SSL support
    - 18.2.5.2. Client-side operation
    - 18.2.5.3. Server-side operation
  - 18.2.6. Notes on non-blocking sockets

    - 18.2.7. Memory BIO Support
    - 18.2.8. SSL session
    - 18.2.9. Security considerations
        - 18.2.9.1. Best defaults
        - 18.2.9.2. Manual settings
            - 18.2.9.2.1. Verifying certificates
            - 18.2.9.2.2. Protocol versions
            - 18.2.9.2.3. Cipher selection
        - 18.2.9.3. Multi-processing
    - 18.2.10. LibreSSL support
- 18.3. `select` — Waiting for I/O completion
    - 18.3.1. `/dev/poll` Polling Objects
    - 18.3.2. Edge and Level Trigger Polling (epoll) Objects
    - 18.3.3. Polling Objects
    - 18.3.4. Kqueue Objects
    - 18.3.5. Kevent Objects
- 18.4. `selectors` — High-level I/O multiplexing
    - 18.4.1. Introduction
    - 18.4.2. Classes
    - 18.4.3. Examples
- 18.5. `asyncio` — Asynchronous I/O, event loop, coroutines and tasks
    - 18.5.1. Base Event Loop
        - 18.5.1.1. Run an event loop
        - 18.5.1.2. Calls
        - 18.5.1.3. Delayed calls
        - 18.5.1.4. Futures
        - 18.5.1.5. Tasks
        - 18.5.1.6. Creating connections
        - 18.5.1.7. Creating listening connections
        - 18.5.1.8. Watch file descriptors
        - 18.5.1.9. Low-level socket operations
        - 18.5.1.10. Resolve host name
        - 18.5.1.11. Connect pipes
        - 18.5.1.12. UNIX signals
        - 18.5.1.13. Executor
        - 18.5.1.14. Error Handling API
        - 18.5.1.15. Debug mode
        - 18.5.1.16. Server
        - 18.5.1.17. Handle
        - 18.5.1.18. Event loop examples
            - 18.5.1.18.1. Hello World with call_soon()
            - 18.5.1.18.2. Display the current date with call_later()
            - 18.5.1.18.3. Watch a file descriptor for read events
            - 18.5.1.18.4. Set signal handlers for SIGINT and SIGTERM
    - 18.5.2. Event loops

        - 18.5.2.1. Event loop functions
        - 18.5.2.2. Available event loops
        - 18.5.2.3. Platform support
            - 18.5.2.3.1. Windows
            - 18.5.2.3.2. Mac OS X
        - 18.5.2.4. Event loop policies and the default policy
        - 18.5.2.5. Event loop policy interface
        - 18.5.2.6. Access to the global loop policy
        - 18.5.2.7. Customizing the event loop policy
    - 18.5.3. Tasks and coroutines
        - 18.5.3.1. Coroutines
            - 18.5.3.1.1. Example: Hello World coroutine
            - 18.5.3.1.2. Example: Coroutine displaying the current date
            - 18.5.3.1.3. Example: Chain coroutines
        - 18.5.3.2. InvalidStateError
        - 18.5.3.3. TimeoutError
        - 18.5.3.4. Future
            - 18.5.3.4.1. Example: Future with run_until_complete()
            - 18.5.3.4.2. Example: Future with run_forever()
        - 18.5.3.5. Task
            - 18.5.3.5.1. Example: Parallel execution of tasks
        - 18.5.3.6. Task functions
    - 18.5.4. Transports and protocols (callback based API)
        - 18.5.4.1. Transports
            - 18.5.4.1.1. BaseTransport
            - 18.5.4.1.2. ReadTransport
            - 18.5.4.1.3. WriteTransport
            - 18.5.4.1.4. DatagramTransport
            - 18.5.4.1.5. BaseSubprocessTransport
        - 18.5.4.2. Protocols
            - 18.5.4.2.1. Protocol classes
            - 18.5.4.2.2. Connection callbacks
            - 18.5.4.2.3. Streaming protocols
            - 18.5.4.2.4. Datagram protocols
            - 18.5.4.2.5. Flow control callbacks
            - 18.5.4.2.6. Coroutines and protocols
        - 18.5.4.3. Protocol examples
            - 18.5.4.3.1. TCP echo client protocol
            - 18.5.4.3.2. TCP echo server protocol
            - 18.5.4.3.3. UDP echo client protocol
            - 18.5.4.3.4. UDP echo server protocol
            - 18.5.4.3.5. Register an open socket to wait for data using a protocol
    - 18.5.5. Streams (coroutine based API)
        - 18.5.5.1. Stream functions

        - 18.5.5.2. StreamReader
        - 18.5.5.3. StreamWriter
        - 18.5.5.4. StreamReaderProtocol
        - 18.5.5.5. IncompleteReadError
        - 18.5.5.6. LimitOverrunError
        - 18.5.5.7. Stream examples
            - 18.5.5.7.1. TCP echo client using streams
            - 18.5.5.7.2. TCP echo server using streams
            - 18.5.5.7.3. Get HTTP headers
            - 18.5.5.7.4. Register an open socket to wait for data using streams
    - 18.5.6. Subprocess
        - 18.5.6.1. Windows event loop
        - 18.5.6.2. Create a subprocess: high-level API using Process
        - 18.5.6.3. Create a subprocess: low-level API using subprocess.Popen
        - 18.5.6.4. Constants
        - 18.5.6.5. Process
        - 18.5.6.6. Subprocess and threads
        - 18.5.6.7. Subprocess examples
            - 18.5.6.7.1. Subprocess using transport and protocol
            - 18.5.6.7.2. Subprocess using streams
    - 18.5.7. Synchronization primitives
        - 18.5.7.1. Locks
            - 18.5.7.1.1. Lock
            - 18.5.7.1.2. Event
            - 18.5.7.1.3. Condition
        - 18.5.7.2. Semaphores
            - 18.5.7.2.1. Semaphore
            - 18.5.7.2.2. BoundedSemaphore
    - 18.5.8. Queues
        - 18.5.8.1. Queue
        - 18.5.8.2. PriorityQueue
        - 18.5.8.3. LifoQueue
            - 18.5.8.3.1. Exceptions
    - 18.5.9. Develop with asyncio
        - 18.5.9.1. Debug mode of asyncio
        - 18.5.9.2. Cancellation
        - 18.5.9.3. Concurrency and multithreading
        - 18.5.9.4. Handle blocking functions correctly
        - 18.5.9.5. Logging
        - 18.5.9.6. Detect coroutine objects never scheduled
        - 18.5.9.7. Detect exceptions never consumed
        - 18.5.9.8. Chain coroutines correctly
        - 18.5.9.9. Pending task destroyed

            - 18.5.9.10. Close transports and event loops
- 18.6. `asyncore` — Asynchronous socket handler
    - 18.6.1. asyncore Example basic HTTP client
    - 18.6.2. asyncore Example basic echo server
- 18.7. `asynchat` — Asynchronous socket command/response handler
    - 18.7.1. asynchat Example
- 18.8. `signal` — Set handlers for asynchronous events
    - 18.8.1. General rules
        - 18.8.1.1. Execution of Python signal handlers
        - 18.8.1.2. Signals and threads
    - 18.8.2. Module contents
    - 18.8.3. Example
- 18.9. `mmap` — Memory-mapped file support