17. Concurrent Execution

The modules described in this chapter provide support for concurrent execution of code. The appropriate choice of tool will depend on the task to be executed (CPU bound vs IO bound) and preferred style of development (event driven cooperative multitasking vs preemptive multitasking). Here's an overview:

- 17.1. `threading` — Thread-based parallelism
  - 17.1.1. Thread-Local Data
  - 17.1.2. Thread Objects
  - 17.1.3. Lock Objects
  - 17.1.4. RLock Objects
  - 17.1.5. Condition Objects
  - 17.1.6. Semaphore Objects
    - 17.1.6.1. `Semaphore` Example
  - 17.1.7. Event Objects
  - 17.1.8. Timer Objects
  - 17.1.9. Barrier Objects
  - 17.1.10. Using locks, conditions, and semaphores in the `with` statement
- 17.2. `multiprocessing` — Process-based parallelism
  - 17.2.1. Introduction
    - 17.2.1.1. The `Process` class
    - 17.2.1.2. Contexts and start methods
    - 17.2.1.3. Exchanging objects between processes
    - 17.2.1.4. Synchronization between processes
    - 17.2.1.5. Sharing state between processes
    - 17.2.1.6. Using a pool of workers
  - 17.2.2. Reference
    - 17.2.2.1. `Process` and exceptions
    - 17.2.2.2. Pipes and Queues
    - 17.2.2.3. Miscellaneous
    - 17.2.2.4. Connection Objects
    - 17.2.2.5. Synchronization primitives
    - 17.2.2.6. Shared `ctypes` Objects
      - 17.2.2.6.1. The `multiprocessing.sharedctypes` module
    - 17.2.2.7. Managers
      - 17.2.2.7.1. Customized managers
      - 17.2.2.7.2. Using a remote manager
    - 17.2.2.8. Proxy Objects
      - 17.2.2.8.1. Cleanup
    - 17.2.2.9. Process Pools
    - 17.2.2.10. Listeners and Clients
      - 17.2.2.10.1. Address Formats
    - 17.2.2.11. Authentication keys

        - 17.2.2.12. Logging
        - 17.2.2.13. The `multiprocessing.dummy` module
    - 17.2.3. Programming guidelines
        - 17.2.3.1. All start methods
        - 17.2.3.2. The *spawn* and *forkserver* start methods
    - 17.2.4. Examples
- 17.3. The `concurrent` package
- 17.4. `concurrent.futures` — Launching parallel tasks
    - 17.4.1. Executor Objects
    - 17.4.2. ThreadPoolExecutor
        - 17.4.2.1. ThreadPoolExecutor Example
    - 17.4.3. ProcessPoolExecutor
        - 17.4.3.1. ProcessPoolExecutor Example
    - 17.4.4. Future Objects
    - 17.4.5. Module Functions
    - 17.4.6. Exception classes
- 17.5. `subprocess` — Subprocess management
    - 17.5.1. Using the `subprocess` Module
        - 17.5.1.1. Frequently Used Arguments
        - 17.5.1.2. Popen Constructor
        - 17.5.1.3. Exceptions
    - 17.5.2. Security Considerations
    - 17.5.3. Popen Objects
    - 17.5.4. Windows Popen Helpers
        - 17.5.4.1. Constants
    - 17.5.5. Older high-level API
    - 17.5.6. Replacing Older Functions with the `subprocess` Module
        - 17.5.6.1. Replacing /bin/sh shell backquote
        - 17.5.6.2. Replacing shell pipeline
        - 17.5.6.3. Replacing `os.system()`
        - 17.5.6.4. Replacing the `os.spawn` family
        - 17.5.6.5. Replacing `os.popen()`, `os.popen2()`, `os.popen3()`
        - 17.5.6.6. Replacing functions from the `popen2` module
    - 17.5.7. Legacy Shell Invocation Functions
    - 17.5.8. Notes
        - 17.5.8.1. Converting an argument sequence to a string on Windows
- 17.6. `sched` — Event scheduler
    - 17.6.1. Scheduler Objects
- 17.7. `queue` — A synchronized queue class
    - 17.7.1. Queue Objects

The following are support modules for some of the above services:

- 17.8. `dummy_threading` — Drop-in replacement for the `threading` module
- 17.9. `_thread` — Low-level threading API

- 17.10. `_dummy_thread` — Drop-in replacement for the `_thread` module