:mod:`multiprocessing` --- Process-based parallelism

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 4)

Unknown directive type "module".

```
.. module:: multiprocessing
   :synopsis: Process-based parallelism.
```

Source code: :source:`Lib/multiprocessing/`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 7); *backlink*

Unknown interpreted text role "source".

Introduction

:mod:`multiprocessing` is a package that supports spawning processes using an API similar to the :mod:`threading` module. The :mod:`multiprocessing` package offers both local and remote concurrency, effectively side-stepping the :term:`Global Interpreter Lock <global interpreter lock>` by using subprocesses instead of threads. Due to this, the :mod:`multiprocessing` module allows the programmer to fully leverage multiple processors on a given machine. It runs on both Unix and Windows.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 14); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 14); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 14); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 14); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 14); *backlink*

Unknown interpreted text role "mod".

The :mod:`multiprocessing` module also introduces APIs which do not have analogs in the :mod:`threading` module. A prime example of this is the :class:`~multiprocessing.pool.Pool` object which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism). The following example demonstrates the common practice of defining such functions in a module so that child processes can successfully import that module. This basic example of data parallelism using :class:`~multiprocessing.pool.Pool`,

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 23); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 23); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 23); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 23); *backlink*

Unknown interpreted text role "class".

```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    with Pool(5) as p:
        print(p.map(f, [1, 2, 3]))
```

will print to standard output

```
[1, 4, 9]
```

The :class:`Process` class

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 46); *backlink*

Unknown interpreted text role "class".

In :mod:`multiprocessing`, processes are spawned by creating a :class:`Process` object and then calling its :meth:`~Process.start` method. :class:`Process` follows the API of :class:`threading.Thread`. A trivial example of a multiprocess program is

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 49); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 49); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 49); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 49); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 49); *backlink*

Unknown interpreted text role "class".

```
from multiprocessing import Process

def f(name):
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
```

```
    p.join()
```

To show the individual process IDs involved, here is an expanded example:

```
from multiprocessing import Process
import os

def info(title):
    print(title)
    print('module name:', __name__)
    print('parent process:', os.getppid())
    print('process id:', os.getpid())

def f(name):
    info('function f')
    print('hello', name)

if __name__ == '__main__':
    info('main line')
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

For an explanation of why the `if __name__ == '__main__'` part is necessary, see :ref:`multiprocessing-programming`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 85); *backlink***
>
> Unknown interpreted text role "ref".

Contexts and start methods

Depending on the platform, :mod:`multiprocessing` supports three ways to start a process. These *start methods* are

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 95); *backlink***
>
> Unknown interpreted text role "mod".

spawn

The parent process starts a fresh python interpreter process. The child process will only inherit those resources necessary to run the process object's :meth:`~Process.run` method. In particular, unnecessary file descriptors and handles from the parent process will not be inherited. Starting a process using this method is rather slow compared to using *fork* or *forkserver*.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 99); *backlink***
>
> Unknown interpreted text role "meth".

Available on Unix and Windows. The default on Windows and macOS.

fork

The parent process uses :func:`os.fork` to fork the Python interpreter. The child process, when it begins, is effectively identical to the parent process. All resources of the parent are inherited by the child process. Note that safely forking a multithreaded process is problematic.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 109); *backlink***
>
> Unknown interpreted text role "func".

Available on Unix only. The default on Unix.

forkserver

When the program starts and selects the *forkserver* start method, a server process is started. From then on, whenever a new process is needed, the parent process connects to the server and requests that it fork a new process. The fork server process is single threaded so it is safe for it to use :func:`os.fork`. No unnecessary resources are inherited.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 118); *backlink*

Unknown interpreted text role "func".

Available on Unix platforms which support passing file descriptors over Unix pipes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 128)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.8

   On macOS, the *spawn* start method is now the default.  The *fork* start
   method should be considered unsafe as it can lead to crashes of the
   subprocess. See :issue:`33725`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 134)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.4
   *spawn* added on all unix platforms, and *forkserver* added for
   some unix platforms.
   Child processes no longer inherit all of the parents inheritable
   handles on Windows.
```

On Unix using the *spawn* or *forkserver* start methods will also start a *resource tracker* process which tracks the unlinked named system resources (such as named semaphores or :class:`~multiprocessing.shared_memory.SharedMemory` objects) created by processes of the program. When all processes have exited the resource tracker unlinks any remaining tracked object. Usually there should be none, but if a process was killed by a signal there may be some "leaked" resources. (Neither leaked semaphores nor shared memory segments will be automatically unlinked until the next reboot. This is problematic for both objects because the system allows only a limited number of named semaphores, and shared memory segments occupy some space in the main memory.)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 140); *backlink*

Unknown interpreted text role "class".

To select a start method you use the :func:`set_start_method` in the `if __name__ == '__main__'` clause of the main module. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 153); *backlink*

Unknown interpreted text role "func".

```
import multiprocessing as mp

def foo(q):
    q.put('hello')

if __name__ == '__main__':
    mp.set_start_method('spawn')
    q = mp.Queue()
    p = mp.Process(target=foo, args=(q,))
    p.start()
    print(q.get())
    p.join()
```

:func:`set_start_method` should not be used more than once in the program.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 170); *backlink*

Unknown interpreted text role "func".

Alternatively, you can use :func:`get_context` to obtain a context object. Context objects have the same API as the multiprocessing module, and allow one to use multiple start methods in the same program.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 173); *backlink*

Unknown interpreted text role "func".

```
import multiprocessing as mp

def foo(q):
    q.put('hello')

if __name__ == '__main__':
    ctx = mp.get_context('spawn')
    q = ctx.Queue()
    p = ctx.Process(target=foo, args=(q,))
    p.start()
    print(q.get())
    p.join()
```

Note that objects related to one context may not be compatible with processes for a different context. In particular, locks created using the *fork* context cannot be passed to processes started using the *spawn* or *forkserver* start methods.

A library which wants to use a particular start method should probably use :func:`get_context` to avoid interfering with the choice of the library user.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 196); *backlink*

Unknown interpreted text role "func".

Warning

The `'spawn'` and `'forkserver'` start methods cannot currently be used with "frozen" executables (i.e., binaries produced by packages like **PyInstaller** and **cx_Freeze**) on Unix. The `'fork'` start method does work.

Exchanging objects between processes

:mod:`multiprocessing` supports two types of communication channel between processes:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 211); *backlink*

Unknown interpreted text role "mod".

Queues

The :class:`Queue` class is a near clone of :class:`queue.Queue`. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 216); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 216); *backlink*

Unknown interpreted text role "class".

```
from multiprocessing import Process, Queue

def f(q):
    q.put([42, None, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=f, args=(q,))
    p.start()
    print(q.get())    # prints "[42, None, 'hello']"
    p.join()
```

Queues are thread and process safe.

Pipes

The :func:`Pipe` function returns a pair of connection objects connected by a pipe which by default is duplex (two-way). For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 235); *backlink*

Unknown interpreted text role "func".

```
from multiprocessing import Process, Pipe

def f(conn):
    conn.send([42, None, 'hello'])
    conn.close()

if __name__ == '__main__':
    parent_conn, child_conn = Pipe()
    p = Process(target=f, args=(child_conn,))
    p.start()
    print(parent_conn.recv())   # prints "[42, None, 'hello']"
    p.join()
```

The two connection objects returned by :func:`Pipe` represent the two ends of the pipe. Each connection object has :meth:`~Connection.send` and :meth:`~Connection.recv` methods (among others). Note that data in a pipe may become corrupted if two processes (or threads) try to read from or write to the *same* end of the pipe at the same time. Of course there is no risk of corruption from processes using different ends of the pipe at the same time.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 251); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 251); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 251); *backlink*

Unknown interpreted text role "meth".

Synchronization between processes

:mod:`multiprocessing` contains equivalents of all the synchronization primitives from :mod:`threading`. For instance one can use a lock to ensure that only one process prints to standard output at a time:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 263); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 263); *backlink*

Unknown interpreted text role "mod".

```
from multiprocessing import Process, Lock

def f(l, i):
    l.acquire()
    try:
        print('hello world', i)
    finally:
        l.release()

if __name__ == '__main__':
    lock = Lock()

    for num in range(10):
```

```
        Process(target=f, args=(lock, num)).start()
```

Without using the lock output from the different processes is liable to get all mixed up.

Sharing state between processes

As mentioned above, when doing concurrent programming it is usually best to avoid using shared state as far as possible. This is particularly true when using multiple processes.

However, if you really do need to use some shared data then :mod:`multiprocessing` provides a couple of ways of doing so.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 293); *backlink*
>
> Unknown interpreted text role "mod".

Shared memory

Data can be stored in a shared memory map using :class:`Value` or :class:`Array`. For example, the following code

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 298); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 298); *backlink*
>
> Unknown interpreted text role "class".

```
from multiprocessing import Process, Value, Array

def f(n, a):
    n.value = 3.1415927
    for i in range(len(a)):
        a[i] = -a[i]

if __name__ == '__main__':
    num = Value('d', 0.0)
    arr = Array('i', range(10))

    p = Process(target=f, args=(num, arr))
    p.start()
    p.join()

    print(num.value)
    print(arr[:])
```

will print

```
3.1415927
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

The `'d'` and `'i'` arguments used when creating `num` and `arr` are typecodes of the kind used by the :mod:`array` module: `'d'` indicates a double precision float and `'i'` indicates a signed integer. These shared objects will be process and thread-safe.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 324); *backlink*
>
> Unknown interpreted text role "mod".

For more flexibility in using shared memory one can use the :mod:`multiprocessing.sharedctypes` module which supports the creation of arbitrary ctypes objects allocated from shared memory.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 329); *backlink*
>
> Unknown interpreted text role "mod".

Server process

A manager object returned by :func:`Manager` controls a server process which holds Python objects and allows other processes to manipulate them using proxies.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 335); *backlink*
>
> Unknown interpreted text role "func".

A manager returned by :func:`Manager` will support types :class:`list`, :class:`dict`, :class:`~managers.Namespace`, :class:`Lock`, :class:`RLock`, :class:`Semaphore`, :class:`BoundedSemaphore`, :class:`Condition`, :class:`Event`, :class:`Barrier`, :class:`Queue`, :class:`Value` and :class:`Array`. For example,

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst`, line 339); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 339); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 339); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 339); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 339); *backlink*

Unknown interpreted text role "class".

```
from multiprocessing import Process, Manager

def f(d, l):
    d[1] = '1'
    d['2'] = 2
    d[0.25] = None
    l.reverse()

if __name__ == '__main__':
    with Manager() as manager:
        d = manager.dict()
        l = manager.list(range(10))

        p = Process(target=f, args=(d, l))
        p.start()
        p.join()

        print(d)
        print(l)
```

will print

```
{0.25: None, 1: '1', '2': 2}
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Server process managers are more flexible than using shared memory objects because they can be made to support arbitrary object types. Also, a single manager can be shared by processes on different computers over a network. They are, however, slower than using shared memory.

Using a pool of workers

The :class:`~multiprocessing.pool.Pool` class represents a pool of worker processes. It has methods which allows tasks to be offloaded to the worker processes in a few different ways.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 379); *backlink*

Unknown interpreted text role "class".

For example:

```
from multiprocessing import Pool, TimeoutError
import time
import os

def f(x):
    return x*x

if __name__ == '__main__':
    # start 4 worker processes
    with Pool(processes=4) as pool:

        # print "[0, 1, 4,..., 81]"
        print(pool.map(f, range(10)))

        # print same numbers in arbitrary order
        for i in pool.imap_unordered(f, range(10)):
            print(i)

        # evaluate "f(20)" asynchronously
        res = pool.apply_async(f, (20,))      # runs in *only* one process
        print(res.get(timeout=1))             # prints "400"

        # evaluate "os.getpid()" asynchronously
        res = pool.apply_async(os.getpid, ()) # runs in *only* one process
        print(res.get(timeout=1))             # prints the PID of that process

        # launching multiple evaluations asynchronously *may* use more processes
        multiple_results = [pool.apply_async(os.getpid, ()) for i in range(4)]
        print([res.get(timeout=1) for res in multiple_results])

        # make a single worker sleep for 10 seconds
        res = pool.apply_async(time.sleep, (10,))
        try:
            print(res.get(timeout=1))
        except TimeoutError:
            print("We lacked patience and got a multiprocessing.TimeoutError")

        print("For the moment, the pool remains available for more work")

    # exiting the 'with'-block has stopped the pool
    print("Now the pool is closed and no longer available")
```

Note that the methods of a pool should only ever be used by the process which created it.

Note

Functionality within this package requires that the `__main__` module be importable by the children. This is covered in :ref:`multiprocessing-programming` however it is worth pointing out here. This means that some examples, such as the :class:`multiprocessing.pool.Pool` examples will not work in the interactive interpreter. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 432); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 432); *backlink*

Unknown interpreted text role "class".

```
>>> from multiprocessing import Pool
>>> p = Pool(5)
>>> def f(x):
...     return x*x
...
>>> with p:
...   p.map(f, [1,2,3])
Process PoolWorker-1:
Process PoolWorker-2:
Process PoolWorker-3:
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
AttributeError: 'module' object has no attribute 'f'
AttributeError: 'module' object has no attribute 'f'
AttributeError: 'module' object has no attribute 'f'
```

(If you try this it will actually output three full tracebacks interleaved in a semi-random fashion, and then you may

have to stop the parent process somehow.)

Reference

The :mod:`multiprocessing` package mostly replicates the API of the :mod:`threading` module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 463); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 463); *backlink*

Unknown interpreted text role "mod".

:class:`Process` and exceptions

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 467); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 470)

Invalid class attribute value for "class" directive: "Process(group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None)".

```
.. class:: Process(group=None, target=None, name=None, args=(), kwargs={}, \
                   *, daemon=None)

   Process objects represent activity that is run in a separate process. The
   :class:`Process` class has equivalents of all the methods of
   :class:`threading.Thread`.

   The constructor should always be called with keyword arguments. *group*
   should always be ``None``; it exists solely for compatibility with
   :class:`threading.Thread`.  *target* is the callable object to be invoked by
   the :meth:`run()` method.  It defaults to ``None``, meaning nothing is
   called. *name* is the process name (see :attr:`name` for more details).
   *args* is the argument tuple for the target invocation.  *kwargs* is a
   dictionary of keyword arguments for the target invocation.  If provided,
   the keyword-only *daemon* argument sets the process :attr:`daemon` flag
   to ``True`` or ``False``.  If ``None`` (the default), this flag will be
   inherited from the creating process.

   By default, no arguments are passed to *target*. The *args* argument,
   which defaults to ``()``, can be used to specify a list or tuple of the arguments
   to pass to *target*.

   If a subclass overrides the constructor, it must make sure it invokes the
   base class constructor (:meth:`Process.__init__`) before doing anything else
   to the process.

   .. versionchanged:: 3.3
      Added the *daemon* argument.

   .. method:: run()

      Method representing the process's activity.

      You may override this method in a subclass.  The standard :meth:`run`
      method invokes the callable object passed to the object's constructor as
      the target argument, if any, with sequential and keyword arguments taken
      from the *args* and *kwargs* arguments, respectively.

      Using a list or tuple as the *args* argument passed to :class:`Process`
      achieves the same effect.

      Example::

         >>> from multiprocessing import Process
         >>> p = Process(target=print, args=[1])
         >>> p.run()
         1
         >>> p = Process(target=print, args=(1,))
```

```
      >>> p.run()
      1

.. method:: start()

   Start the process's activity.

   This must be called at most once per process object.  It arranges for the
   object's :meth:`run` method to be invoked in a separate process.

.. method:: join([timeout])

   If the optional argument *timeout* is ``None`` (the default), the method
   blocks until the process whose :meth:`join` method is called terminates.
   If *timeout* is a positive number, it blocks at most *timeout* seconds.
   Note that the method returns ``None`` if its process terminates or if the
   method times out.  Check the process's :attr:`exitcode` to determine if
   it terminated.

   A process can be joined many times.

   A process cannot join itself because this would cause a deadlock.  It is
   an error to attempt to join a process before it has been started.

.. attribute:: name

   The process's name.  The name is a string used for identification purposes
   only.  It has no semantics.  Multiple processes may be given the same
   name.

   The initial name is set by the constructor.  If no explicit name is
   provided to the constructor, a name of the form
   'Process-N\ :sub:`1`:N\ :sub:`2`:...:N\ :sub:`k`' is constructed, where
   each N\ :sub:`k` is the N-th child of its parent.

.. method:: is_alive

   Return whether the process is alive.

   Roughly, a process object is alive from the moment the :meth:`start`
   method returns until the child process terminates.

.. attribute:: daemon

   The process's daemon flag, a Boolean value.  This must be set before
   :meth:`start` is called.

   The initial value is inherited from the creating process.

   When a process exits, it attempts to terminate all of its daemonic child
   processes.

   Note that a daemonic process is not allowed to create child processes.
   Otherwise a daemonic process would leave its children orphaned if it gets
   terminated when its parent process exits. Additionally, these are **not**
   Unix daemons or services, they are normal processes that will be
   terminated (and not joined) if non-daemonic processes have exited.

In addition to the  :class:`threading.Thread` API, :class:`Process` objects
also support the following attributes and methods:

.. attribute:: pid

   Return the process ID.  Before the process is spawned, this will be
   ``None``.

.. attribute:: exitcode

   The child's exit code.  This will be ``None`` if the process has not yet
   terminated.

   If the child's :meth:`run` method returned normally, the exit code
   will be 0.  If it terminated via :func:`sys.exit` with an integer
   argument *N*, the exit code will be *N*.

   If the child terminated due to an exception not caught within
   :meth:`run`, the exit code will be 1.  If it was terminated by
   signal *N*, the exit code will be the negative value *-N*.

.. attribute:: authkey

   The process's authentication key (a byte string).

   When :mod:`multiprocessing` is initialized the main process is assigned a
```

```
      random string using :func:`os.urandom`.

      When a :class:`Process` object is created, it will inherit the
      authentication key of its parent process, although this may be changed by
      setting :attr:`authkey` to another byte string.

      See :ref:`multiprocessing-auth-keys`.

   .. attribute:: sentinel

      A numeric handle of a system object which will become "ready" when
      the process ends.

      You can use this value if you want to wait on several events at
      once using :func:`multiprocessing.connection.wait`.  Otherwise
      calling :meth:`join()` is simpler.

      On Windows, this is an OS handle usable with the ``WaitForSingleObject``
      and ``WaitForMultipleObjects`` family of API calls.  On Unix, this is
      a file descriptor usable with primitives from the :mod:`select` module.

      .. versionadded:: 3.3

   .. method:: terminate()

      Terminate the process.  On Unix this is done using the ``SIGTERM`` signal;
      on Windows :c:func:`TerminateProcess` is used.  Note that exit handlers and
      finally clauses, etc., will not be executed.

      Note that descendant processes of the process will *not* be terminated --
      they will simply become orphaned.

      .. warning::

         If this method is used when the associated process is using a pipe or
         queue then the pipe or queue is liable to become corrupted and may
         become unusable by other process.  Similarly, if the process has
         acquired a lock or semaphore etc. then terminating it is liable to
         cause other processes to deadlock.

   .. method:: kill()

      Same as :meth:`terminate()` but using the ``SIGKILL`` signal on Unix.

      .. versionadded:: 3.7

   .. method:: close()

      Close the :class:`Process` object, releasing all resources associated
      with it.  :exc:`ValueError` is raised if the underlying process
      is still running.  Once :meth:`close` returns successfully, most
      other methods and attributes of the :class:`Process` object will
      raise :exc:`ValueError`.

      .. versionadded:: 3.7

   Note that the :meth:`start`, :meth:`join`, :meth:`is_alive`,
   :meth:`terminate` and :attr:`exitcode` methods should only be called by
   the process that created the process object.

   Example usage of some of the methods of :class:`Process`:

   .. doctest::
      :options: +ELLIPSIS

       >>> import multiprocessing, time, signal
       >>> p = multiprocessing.Process(target=time.sleep, args=(1000,))
       >>> print(p, p.is_alive())
       <Process ... initial> False
       >>> p.start()
       >>> print(p, p.is_alive())
       <Process ... started> True
       >>> p.terminate()
       >>> time.sleep(0.1)
       >>> print(p, p.is_alive())
       <Process ... stopped exitcode=-SIGTERM> False
       >>> p.exitcode == -signal.SIGTERM
       True
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 681)

Unknown directive type "exception".

```
.. exception:: ProcessError

   The base class of all :mod:`multiprocessing` exceptions.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 685)

Unknown directive type "exception".

```
.. exception:: BufferTooShort

   Exception raised by :meth:`Connection.recv_bytes_into()` when the supplied
   buffer object is too small for the message read.

   If ``e`` is an instance of :exc:`BufferTooShort` then ``e.args[0]`` will give
   the message as a byte string.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 693)

Unknown directive type "exception".

```
.. exception:: AuthenticationError

   Raised when there is an authentication error.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 697)

Unknown directive type "exception".

```
.. exception:: TimeoutError

   Raised by methods with a timeout when the timeout expires.
```

Pipes and Queues

When using multiple processes, one generally uses message passing for communication between processes and avoids having to use any synchronization primitives like locks.

For passing messages one can use :func:`Pipe` (for a connection between two processes) or a queue (which allows multiple producers and consumers).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 708); *backlink*

Unknown interpreted text role "func".

The :class:`Queue`, :class:`SimpleQueue` and :class:`JoinableQueue` types are multi-producer, multi-consumer :abbr:`FIFO (first-in, first-out)` queues modelled on the :class:`queue.Queue` class in the standard library. They differ in that :class:`Queue` lacks the :meth:`~queue.Queue.task_done` and :meth:`~queue.Queue.join` methods introduced into Python 2.5's :class:`queue.Queue` class.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "abbr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 711); *backlink*

Unknown interpreted text role "class".

If you use :class:`JoinableQueue` then you **must** call :meth:`JoinableQueue.task_done` for each task removed from the queue or else the semaphore used to count the number of unfinished tasks may eventually overflow, raising an exception.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 718); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 718); *backlink*

Unknown interpreted text role "meth".

Note that one can also create a shared queue by using a manager object -- see :ref:`multiprocessing-managers`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 723); *backlink*

Unknown interpreted text role "ref".

Note

:mod:`multiprocessing` uses the usual :exc:`queue.Empty` and :exc:`queue.Full` exceptions to signal a timeout. They are not available in the :mod:`multiprocessing` namespace so you need to import them from :mod:`queue`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 728); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 728); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 728); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 728); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 728); *backlink*

Unknown interpreted text role "mod".

Note

When an object is put on a queue, the object is pickled and a background thread later flushes the pickled data to an underlying pipe. This has some consequences which are a little surprising, but should not cause any practical difficulties -- if they really bother you then you can instead use a queue created with a :ref:`manager <multiprocessing-managers>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 735); *backlink*

Unknown interpreted text role "ref".

1. After putting an object on an empty queue there may be an infinitesimal delay before the queue's :meth:`~Queue.empty` method returns :const:`False` and :meth:`~Queue.get_nowait` can return without raising :exc:`queue.Empty`.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 742); *backlink*

   Unknown interpreted text role "meth".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 742); *backlink*

   Unknown interpreted text role "const".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 742); *backlink*

   Unknown interpreted text role "meth".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 742); *backlink*

   Unknown interpreted text role "exc".

2. If multiple processes are enqueuing objects, it is possible for the objects to be received at the other end out-of-order. However, objects enqueued by the same process will always be in the expected order with respect to each other.

Warning

If a process is killed using :meth:`Process.terminate` or :func:`os.kill` while it is trying to use a :class:`Queue`, then the

data in the queue is likely to become corrupted. This may cause any other process to get an exception when it tries to use the queue later on.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 754); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 754); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 754); *backlink*

Unknown interpreted text role "class".

Warning

As mentioned above, if a child process has put items on a queue (and it has not used :meth:`JoinableQueue.cancel_join_thread <multiprocessing.Queue.cancel_join_thread>`), then that process will not terminate until all buffered items have been flushed to the pipe.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 761); *backlink*

Unknown interpreted text role "meth".

This means that if you try joining that process you may get a deadlock unless you are sure that all items which have been put on the queue have been consumed. Similarly, if the child process is non-daemonic then the parent process may hang on exit when it tries to join all its non-daemonic children.

Note that a queue created using a manager does not have this issue. See :ref:`multiprocessing-programming`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 771); *backlink*

Unknown interpreted text role "ref".

For an example of the usage of queues for interprocess communication see :ref:`multiprocessing-examples`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 774); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 778)

Unknown directive type "function".

```
.. function:: Pipe([duplex])

   Returns a pair ``(conn1, conn2)`` of
   :class:`~multiprocessing.connection.Connection` objects representing the
   ends of a pipe.

   If *duplex* is ``True`` (the default) then the pipe is bidirectional.  If
   *duplex* is ``False`` then the pipe is unidirectional: ``conn1`` can only be
   used for receiving messages and ``conn2`` can only be used for sending
   messages.
```

Returns a process shared queue implemented using a pipe and a few locks/semaphores. When a process first puts an item on the queue a feeder thread is started which transfers objects from a buffer into the pipe.

The usual :exc:`queue.Empty` and :exc:`queue.Full` exceptions from the standard library's :mod:`queue` module are raised to signal timeouts.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 796); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 796); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 796); *backlink*

Unknown interpreted text role "mod".

:class:`Queue` implements all the methods of :class:`queue.Queue` except for :meth:`~queue.Queue.task_done` and :meth:`~queue.Queue.join`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 799); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 799); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 799); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 799); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 802)

Unknown directive type "method".

```
.. method:: qsize()

   Return the approximate size of the queue.  Because of
   multithreading/multiprocessing semantics, this number is not reliable.

   Note that this may raise :exc:`NotImplementedError` on Unix platforms like
   macOS where ``sem_getvalue()`` is not implemented.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 810)

Unknown directive type "method".

```
.. method:: empty()

   Return ``True`` if the queue is empty, ``False`` otherwise.  Because of
   multithreading/multiprocessing semantics, this is not reliable.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 815)

Unknown directive type "method".

```
.. method:: full()

   Return ``True`` if the queue is full, ``False`` otherwise.  Because of
   multithreading/multiprocessing semantics, this is not reliable.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 820)

Unknown directive type "method".

```
.. method:: put(obj[, block[, timeout]])

   Put obj into the queue.  If the optional argument *block* is ``True``
   (the default) and *timeout* is ``None`` (the default), block if necessary until
   a free slot is available.  If *timeout* is a positive number, it blocks at
   most *timeout* seconds and raises the :exc:`queue.Full` exception if no
   free slot was available within that time.  Otherwise (*block* is
   ``False``), put an item on the queue if a free slot is immediately
   available, else raise the :exc:`queue.Full` exception (*timeout* is
   ignored in that case).

   .. versionchanged:: 3.8
      If the queue is closed, :exc:`ValueError` is raised instead of
      :exc:`AssertionError`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 835)

Unknown directive type "method".

```
.. method:: put_nowait(obj)

   Equivalent to ``put(obj, False)``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 839)

Unknown directive type "method".

```
.. method:: get([block[, timeout]])

   Remove and return an item from the queue.  If optional args *block* is
   ``True`` (the default) and *timeout* is ``None`` (the default), block if
   necessary until an item is available.  If *timeout* is a positive number,
   it blocks at most *timeout* seconds and raises the :exc:`queue.Empty`
   exception if no item was available within that time.  Otherwise (block is
   ``False``), return an item if one is immediately available, else raise the
   :exc:`queue.Empty` exception (*timeout* is ignored in that case).

   .. versionchanged:: 3.8
      If the queue is closed, :exc:`ValueError` is raised instead of
      :exc:`OSError`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 853)

Unknown directive type "method".

```
.. method:: get_nowait()

   Equivalent to ``get(False)``.
```

:class:`multiprocessing.Queue` has a few additional methods not found in :class:`queue.Queue`. These methods are usually unnecessary for most code:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 857); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 857); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 861)

Unknown directive type "method".

```
.. method:: close()

   Indicate that no more data will be put on this queue by the current
   process.  The background thread will quit once it has flushed all buffered
   data to the pipe.  This is called automatically when the queue is garbage
   collected.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 868)

Unknown directive type "method".

```
.. method:: join_thread()

   Join the background thread.  This can only be used after :meth:`close` has
   been called.  It blocks until the background thread exits, ensuring that
   all data in the buffer has been flushed to the pipe.

   By default if a process is not the creator of the queue then on exit it
   will attempt to join the queue's background thread.  The process can call
   :meth:`cancel_join_thread` to make :meth:`join_thread` do nothing.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 878)

Unknown directive type "method".

```
.. method:: cancel_join_thread()

   Prevent :meth:`join_thread` from blocking.  In particular, this prevents
   the background thread from being joined automatically when the process
   exits -- see :meth:`join_thread`.

   A better name for this method might be
   ``allow_exit_without_flush()``.  It is likely to cause enqueued
   data to be lost, and you almost certainly will not need to use it.
   It is really only there if you need the current process to exit
   immediately without waiting to flush enqueued data to the
   underlying pipe, and you don't care about lost data.
```

Note

This class's functionality requires a functioning shared semaphore implementation on the host operating system. Without one, the functionality in this class will be disabled, and attempts to instantiate a :class:`Queue` will result in an :exc:`ImportError`. See :issue:`3770` for additional information. The same holds true for any of the specialized queue types listed below.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 893); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 893); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 893); *backlink*

Unknown interpreted text role "issue".

It is a simplified :class:`Queue` type, very close to a locked :class:`Pipe`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 902); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 902); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 904)

Unknown directive type "method".

```
.. method:: close()

   Close the queue: release internal resources.

   A queue must not be used anymore after it is closed. For example,
   :meth:`get`, :meth:`put` and :meth:`empty` methods must no longer be
   called.

   .. versionadded:: 3.9
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 914)

Unknown directive type "method".

```
.. method:: empty()

   Return ``True`` if the queue is empty, ``False`` otherwise.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 918)

Unknown directive type "method".

```
.. method:: get()

   Remove and return an item from the queue.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 922)

Unknown directive type "method".

```
.. method:: put(item)

   Put *item* into the queue.
```

:class:`JoinableQueue`, a :class:`Queue` subclass, is a queue which additionally has :meth:`task_done` and :meth:`join` methods.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 929); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 929); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 929); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 929); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 932)

Unknown directive type "method".

```
.. method:: task_done()

   Indicate that a formerly enqueued task is complete. Used by queue
   consumers.  For each :meth:`~Queue.get` used to fetch a task, a subsequent
   call to :meth:`task_done` tells the queue that the processing on the task
   is complete.

   If a :meth:`~queue.Queue.join` is currently blocking, it will resume when all
   items have been processed (meaning that a :meth:`task_done` call was
   received for every item that had been :meth:`~Queue.put` into the queue).

   Raises a :exc:`ValueError` if called more times than there were items
   placed in the queue.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 947)

Unknown directive type "method".

```
.. method:: join()

   Block until all items in the queue have been gotten and processed.

   The count of unfinished tasks goes up whenever an item is added to the
   queue.  The count goes down whenever a consumer calls
   :meth:`task_done` to indicate that the item was retrieved and all work on
   it is complete.  When the count of unfinished tasks drops to zero,
   :meth:`~queue.Queue.join` unblocks.
```

Miscellaneous

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 961)

Unknown directive type "function".

```
.. function:: active_children()

   Return list of all live children of the current process.

   Calling this has the side effect of "joining" any processes which have
   already finished.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 968)

Unknown directive type "function".

```
.. function:: cpu_count()

   Return the number of CPUs in the system.

   This number is not equivalent to the number of CPUs the current process can
   use.  The number of usable CPUs can be obtained with
   ``len(os.sched_getaffinity(0))``

   When the number of CPUs cannot be determined a :exc:`NotImplementedError`
```

```
   is raised.

   .. seealso::
      :func:`os.cpu_count`
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 982)

Unknown directive type "function".

```
.. function:: current_process()

   Return the :class:`Process` object corresponding to the current process.

   An analogue of :func:`threading.current_thread`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 988)

Unknown directive type "function".

```
.. function:: parent_process()

   Return the :class:`Process` object corresponding to the parent process of
   the :func:`current_process`. For the main process, ``parent_process`` will
   be ``None``.

   .. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 996)

Unknown directive type "function".

```
.. function:: freeze_support()

   Add support for when a program which uses :mod:`multiprocessing` has been
   frozen to produce a Windows executable.  (Has been tested with **py2exe**,
   **PyInstaller** and **cx_Freeze**.)

   One needs to call this function straight after the ``if __name__ ==
   '__main__'`` line of the main module.  For example::

      from multiprocessing import Process, freeze_support

      def f():
          print('hello world!')

      if __name__ == '__main__':
          freeze_support()
          Process(target=f).start()

   If the ``freeze_support()`` line is omitted then trying to run the frozen
   executable will raise :exc:`RuntimeError`.

   Calling ``freeze_support()`` has no effect when invoked on any operating
   system other than Windows.  In addition, if the module is being run
   normally by the Python interpreter on Windows (the program has not been
   frozen), then ``freeze_support()`` has no effect.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1022)

Unknown directive type "function".

```
.. function:: get_all_start_methods()

   Returns a list of the supported start methods, the first of which
   is the default.  The possible start methods are ``'fork'``,
   ``'spawn'`` and ``'forkserver'``.  On Windows only ``'spawn'`` is
   available.  On Unix ``'fork'`` and ``'spawn'`` are always
   supported, with ``'fork'`` being the default.

   .. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1032)

Unknown directive type "function".

```
.. function:: get_context(method=None)

   Return a context object which has the same attributes as the
   :mod:`multiprocessing` module.

   If *method* is ``None`` then the default context is returned.
   Otherwise *method* should be ``'fork'``, ``'spawn'``,
   ``'forkserver'``.  :exc:`ValueError` is raised if the specified
   start method is not available.

   .. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1044)

Unknown directive type "function".

```
.. function:: get_start_method(allow_none=False)

   Return the name of start method used for starting processes.

   If the start method has not been fixed and *allow_none* is false,
   then the start method is fixed to the default and the name is
   returned.  If the start method has not been fixed and *allow_none*
   is true then ``None`` is returned.

   The return value can be ``'fork'``, ``'spawn'``, ``'forkserver'``
   or ``None``.  ``'fork'`` is the default on Unix, while ``'spawn'`` is
   the default on Windows and macOS.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1057)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.8

   On macOS, the *spawn* start method is now the default.  The *fork* start
   method should be considered unsafe as it can lead to crashes of the
   subprocess. See :issue:`33725`.

   .. versionadded:: 3.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1065)

Unknown directive type "function".

```
.. function:: set_executable(executable)

   Set the path of the Python interpreter to use when starting a child process.
   (By default :data:`sys.executable` is used).  Embedders will probably need to
   do some thing like ::

      set_executable(os.path.join(sys.exec_prefix, 'pythonw.exe'))

   before they can create child processes.

   .. versionchanged:: 3.4
      Now supported on Unix when the ``'spawn'`` start method is used.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1078)

Unknown directive type "function".

```
.. function:: set_start_method(method)

   Set the method which should be used to start child processes.
   *method* can be ``'fork'``, ``'spawn'`` or ``'forkserver'``.
```

```
Note that this should be called at most once, and it should be
protected inside the ``if __name__ == '__main__'`` clause of the
main module.

.. versionadded:: 3.4
```

Note

:mod:`multiprocessing` contains no analogues of :func:`threading.active_count`, :func:`threading.enumerate`, :func:`threading.settrace`, :func:`threading.setprofile`, :class:`threading.Timer`, or :class:`threading.local`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst`, line 1091); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst`, line 1091); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst`, line 1091); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst`, line 1091); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst`, line 1091); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst`, line 1091); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst`, line 1091); *backlink*

Unknown interpreted text role "class".

Connection Objects

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst`, line 1100)

Unknown directive type "currentmodule".

```
.. currentmodule:: multiprocessing.connection
```

Connection objects allow the sending and receiving of picklable objects or strings. They can be thought of as message oriented connected sockets.

Connection objects are usually created using :func:`Pipe <multiprocessing.Pipe>` -- see also :ref:`multiprocessing-listeners-clients`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1105); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1105); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1111)

Unknown directive type "method".

```
.. method:: send(obj)

   Send an object to the other end of the connection which should be read
   using :meth:`recv`.

   The object must be picklable.  Very large pickles (approximately 32 MiB+,
   though it depends on the OS) may raise a :exc:`ValueError` exception.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1119)

Unknown directive type "method".

```
.. method:: recv()

   Return an object sent from the other end of the connection using
   :meth:`send`.  Blocks until there is something to receive.  Raises
   :exc:`EOFError` if there is nothing left to receive
   and the other end was closed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1126)

Unknown directive type "method".

```
.. method:: fileno()

   Return the file descriptor or handle used by the connection.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1130)

Unknown directive type "method".

```
.. method:: close()

   Close the connection.

   This is called automatically when the connection is garbage collected.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1136)

Unknown directive type "method".

```
.. method:: poll([timeout])

   Return whether there is any data available to be read.

   If *timeout* is not specified then it will return immediately.  If
   *timeout* is a number then this specifies the maximum time in seconds to
   block.  If *timeout* is ``None`` then an infinite timeout is used.

   Note that multiple connection objects may be polled at once by
   using :func:`multiprocessing.connection.wait`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1147)

Unknown directive type "method".

```
.. method:: send_bytes(buffer[, offset[, size]])

   Send byte data from a :term:`bytes-like object` as a complete message.

   If *offset* is given then data is read from that position in *buffer*.  If
   *size* is given then that many bytes will be read from buffer.  Very large
   buffers (approximately 32 MiB+, though it depends on the OS) may raise a
   :exc:`ValueError` exception
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1156)

Unknown directive type "method".

```
.. method:: recv_bytes([maxlength])

   Return a complete message of byte data sent from the other end of the
   connection as a string.  Blocks until there is something to receive.
   Raises :exc:`EOFError` if there is nothing left
   to receive and the other end has closed.

   If *maxlength* is specified and the message is longer than *maxlength*
   then :exc:`OSError` is raised and the connection will no longer be
   readable.

   .. versionchanged:: 3.3
      This function used to raise :exc:`IOError`, which is now an
      alias of :exc:`OSError`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1172)

Unknown directive type "method".

```
.. method:: recv_bytes_into(buffer[, offset])

   Read into *buffer* a complete message of byte data sent from the other end
   of the connection and return the number of bytes in the message.  Blocks
   until there is something to receive.  Raises
   :exc:`EOFError` if there is nothing left to receive and the other end was
   closed.

   *buffer* must be a writable :term:`bytes-like object`.  If
   *offset* is given then the message will be written into the buffer from
   that position.  Offset must be a non-negative integer less than the
   length of *buffer* (in bytes).

   If the buffer is too short then a :exc:`BufferTooShort` exception is
   raised and the complete message is available as ``e.args[0]`` where ``e``
   is the exception instance.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1189)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   Connection objects themselves can now be transferred between processes
   using :meth:`Connection.send` and :meth:`Connection.recv`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1193)

Unknown directive type "versionadded".

```
.. versionadded:: 3.3
   Connection objects now support the context management protocol -- see
   :ref:`typecontextmanager`.  :meth:`~contextmanager.__enter__` returns the
   connection object, and :meth:`~contextmanager.__exit__` calls :meth:`close`.
```

For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1200)

Unknown directive type "doctest".

```
.. doctest::

    >>> from multiprocessing import Pipe
    >>> a, b = Pipe()
    >>> a.send([1, 'hello', None])
    >>> b.recv()
    [1, 'hello', None]
    >>> b.send_bytes(b'thank you')
    >>> a.recv_bytes()
    b'thank you'
    >>> import array
    >>> arr1 = array.array('i', range(5))
    >>> arr2 = array.array('i', [0] * 10)
    >>> a.send_bytes(arr1)
    >>> count = b.recv_bytes_into(arr2)
    >>> assert count == len(arr1) * arr1.itemsize
    >>> arr2
    array('i', [0, 1, 2, 3, 4, 0, 0, 0, 0, 0])
```

Warning

The :meth:`Connection.recv` method automatically unpickles the data it receives, which can be a security risk unless you can trust the process which sent the message.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1223); *backlink*

Unknown interpreted text role "meth".

Therefore, unless the connection object was produced using :func:`Pipe` you should only use the :meth:`~Connection.recv` and :meth:`~Connection.send` methods after performing some sort of authentication. See :ref:`multiprocessing-auth-keys`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1227); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1227); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1227); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1227); *backlink*

Unknown interpreted text role "ref".

Warning

If a process is killed while it is trying to read or write to a pipe then the data in the pipe is likely to become corrupted, because it may become impossible to be sure where the message boundaries lie.

Synchronization primitives

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1242)

Unknown directive type "currentmodule".

```
.. currentmodule:: multiprocessing
```

Generally synchronization primitives are not as necessary in a multiprocess program as they are in a multithreaded program. See the documentation for :mod:`threading` module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1244); *backlink*

Unknown interpreted text role "mod".

Note that one can also create synchronization primitives by using a manager object -- see :ref:`multiprocessing-managers`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1248); *backlink*

Unknown interpreted text role "ref".

A barrier object: a clone of :class:`threading.Barrier`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1253); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1255)

Unknown directive type "versionadded".

```
.. versionadded:: 3.3
```

A bounded semaphore object: a close analog of :class:`threading.BoundedSemaphore`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1259); *backlink*

Unknown interpreted text role "class".

A solitary difference from its close analog exists: its `acquire` method's first argument is named *block*, as is consistent with :meth:`Lock.acquire`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1262); *backlink*

Unknown interpreted text role "meth".

Note

On macOS, this is indistinguishable from :class:`Semaphore` because `sem_getvalue()` is not implemented on that platform.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1266); *backlink*

Unknown interpreted text role "class".

A condition variable: an alias for :class:`threading.Condition`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1271); *backlink*

Unknown interpreted text role "class".

If *lock* is specified then it should be a :class:`Lock` or :class:`RLock` object from :mod:`multiprocessing`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1273); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1273); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1273); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1276)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   The :meth:`~threading.Condition.wait_for` method was added.
```

A clone of :class:`threading.Event`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1281); *backlink*

Unknown interpreted text role "class".

A non-recursive lock object: a close analog of :class:`threading.Lock`. Once a process or thread has acquired a lock, subsequent attempts to acquire it from any process or thread will block until it is released; any process or thread may release it. The concepts and behaviors of :class:`threading.Lock` as it applies to threads are replicated here in :class:`multiprocessing.Lock` as it applies to either processes or threads, except as noted.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1286); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1286); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1286); *backlink*

Unknown interpreted text role "class".

Note that :class:`Lock` is actually a factory function which returns an instance of `multiprocessing.synchronize.Lock` initialized with a default context.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1294); *backlink*

Unknown interpreted text role "class".

:class:`Lock` supports the :term:`context manager` protocol and thus may be used in :keyword:`with` statements.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1298); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1298); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1298); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1301)

Unknown directive type "method".

```
.. method:: acquire(block=True, timeout=None)

   Acquire a lock, blocking or non-blocking.

   With the *block* argument set to ``True`` (the default), the method call
   will block until the lock is in an unlocked state, then set it to locked
   and return ``True``.  Note that the name of this first argument differs
   from that in :meth:`threading.Lock.acquire`.

   With the *block* argument set to ``False``, the method call does not
   block.  If the lock is currently in a locked state, return ``False``;
   otherwise set the lock to a locked state and return ``True``.

   When invoked with a positive, floating-point value for *timeout*, block
   for at most the number of seconds specified by *timeout* as long as
   the lock can not be acquired.  Invocations with a negative value for
   *timeout* are equivalent to a *timeout* of zero.  Invocations with a
   *timeout* value of ``None`` (the default) set the timeout period to
   infinite.  Note that the treatment of negative or ``None`` values for
   *timeout* differs from the implemented behavior in
   :meth:`threading.Lock.acquire`.  The *timeout* argument has no practical
   implications if the *block* argument is set to ``False`` and is thus
   ignored.  Returns ``True`` if the lock has been acquired or ``False`` if
   the timeout period has elapsed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1327)

Unknown directive type "method".

```
.. method:: release()

   Release a lock.  This can be called from any process or thread, not only
   the process or thread which originally acquired the lock.

   Behavior is the same as in :meth:`threading.Lock.release` except that
   when invoked on an unlocked lock, a :exc:`ValueError` is raised.
```

A recursive lock object: a close analog of :class:`threading.RLock`. A recursive lock must be released by the process or thread that acquired it. Once a process or thread has acquired a recursive lock, the same process or thread may acquire it again without blocking; that process or thread must release it once for each time it has been acquired.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1338); *backlink*

Unknown interpreted text role "class".

Note that :class:`RLock` is actually a factory function which returns an instance of `multiprocessing.synchronize.RLock` initialized with a default context.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1344); *backlink*

Unknown interpreted text role "class".

:class:`RLock` supports the :term:`context manager` protocol and thus may be used in :keyword:`with` statements.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1348); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1348); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1348); *backlink*

Unknown interpreted text role "keyword".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1352)

Unknown directive type "method".

```
.. method:: acquire(block=True, timeout=None)

   Acquire a lock, blocking or non-blocking.

   When invoked with the *block* argument set to ``True``, block until the
   lock is in an unlocked state (not owned by any process or thread) unless
   the lock is already owned by the current process or thread.  The current
   process or thread then takes ownership of the lock (if it does not
   already have ownership) and the recursion level inside the lock increments
   by one, resulting in a return value of ``True``.  Note that there are
   several differences in this first argument's behavior compared to the
   implementation of :meth:`threading.RLock.acquire`, starting with the name
   of the argument itself.

   When invoked with the *block* argument set to ``False``, do not block.
   If the lock has already been acquired (and thus is owned) by another
   process or thread, the current process or thread does not take ownership
   and the recursion level within the lock is not changed, resulting in
   a return value of ``False``.  If the lock is in an unlocked state, the
   current process or thread takes ownership and the recursion level is
   incremented, resulting in a return value of ``True``.

   Use and behaviors of the *timeout* argument are the same as in
   :meth:`Lock.acquire`.  Note that some of these behaviors of *timeout*
   differ from the implemented behaviors in :meth:`threading.RLock.acquire`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1379)

Unknown directive type "method".

```
.. method:: release()

   Release a lock, decrementing the recursion level.  If after the
   decrement the recursion level is zero, reset the lock to unlocked (not
   owned by any process or thread) and if any other processes or threads
   are blocked waiting for the lock to become unlocked, allow exactly one
   of them to proceed.  If after the decrement the recursion level is still
   nonzero, the lock remains locked and owned by the calling process or
   thread.

   Only call this method when the calling process or thread owns the lock.
   An :exc:`AssertionError` is raised if this method is called by a process
   or thread other than the owner or if the lock is in an unlocked (unowned)
   state.  Note that the type of exception raised in this situation
   differs from the implemented behavior in :meth:`threading.RLock.release`.
```

A semaphore object: a close analog of :class:`threading.Semaphore`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1398); *backlink*

Unknown interpreted text role "class".

A solitary difference from its close analog exists: its `acquire` method's first argument is named *block*, as is consistent with :meth:`Lock.acquire`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1400); *backlink*

Unknown interpreted text role "meth".

Note

On macOS, `sem_timedwait` is unsupported, so calling `acquire()` with a timeout will emulate that function's behavior using a sleeping loop.

Note

If the SIGINT signal generated by :kbd:`Ctrl-C` arrives while the main thread is blocked by a call to :meth:`BoundedSemaphore.acquire`, :meth:`Lock.acquire`, :meth:`RLock.acquire`, :meth:`Semaphore.acquire`, :meth:`Condition.acquire` or :meth:`Condition.wait` then the call will be immediately interrupted and :exc:`KeyboardInterrupt` will be raised.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1410); *backlink*

Unknown interpreted text role "kbd".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1410); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1410); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1410); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1410); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1410); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 1410); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1410); *backlink*

Unknown interpreted text role "exc".

This differs from the behaviour of :mod:`threading` where SIGINT will be ignored while the equivalent blocking calls are in progress.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1416); *backlink*

Unknown interpreted text role "mod".

Note

Some of this package's functionality requires a functioning shared semaphore implementation on the host operating system. Without one, the :mod:`multiprocessing.synchronize` module will be disabled, and attempts to import it will result in an :exc:`ImportError`. See :issue:`3770` for additional information.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1421); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1421); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1421); *backlink*

Unknown interpreted text role "issue".

Shared :mod:`ctypes` Objects

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1428); *backlink*

Unknown interpreted text role "mod".

It is possible to create shared objects using shared memory which can be inherited by child processes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1434)

Unknown directive type "function".

```
.. function:: Value(typecode_or_type, *args, lock=True)

   Return a :mod:`ctypes` object allocated from shared memory.  By default the
   return value is actually a synchronized wrapper for the object.  The object
   itself can be accessed via the *value* attribute of a :class:`Value`.

   *typecode_or_type* determines the type of the returned object: it is either a
   ctypes type or a one character typecode of the kind used by the :mod:`array`
   module.  *\*args* is passed on to the constructor for the type.

   If *lock* is ``True`` (the default) then a new recursive lock
   object is created to synchronize access to the value.  If *lock* is
   a :class:`Lock` or :class:`RLock` object then that will be used to
   synchronize access to the value.  If *lock* is ``False`` then
   access to the returned object will not be automatically protected
   by a lock, so it will not necessarily be "process-safe".
```

```
Operations like ``+=`` which involve a read and write are not
atomic.  So if, for instance, you want to atomically increment a
shared value it is insufficient to just do ::

    counter.value += 1

Assuming the associated lock is recursive (which it is by default)
you can instead do ::

    with counter.get_lock():
        counter.value += 1

Note that *lock* is a keyword-only argument.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1465)

Unknown directive type "function".

```
.. function:: Array(typecode_or_type, size_or_initializer, *, lock=True)

   Return a ctypes array allocated from shared memory.  By default the return
   value is actually a synchronized wrapper for the array.

   *typecode_or_type* determines the type of the elements of the returned array:
   it is either a ctypes type or a one character typecode of the kind used by
   the :mod:`array` module.  If *size_or_initializer* is an integer, then it
   determines the length of the array, and the array will be initially zeroed.
   Otherwise, *size_or_initializer* is a sequence which is used to initialize
   the array and whose length determines the length of the array.

   If *lock* is ``True`` (the default) then a new lock object is created to
   synchronize access to the value.  If *lock* is a :class:`Lock` or
   :class:`RLock` object then that will be used to synchronize access to the
   value.  If *lock* is ``False`` then access to the returned object will not be
   automatically protected by a lock, so it will not necessarily be
   "process-safe".

   Note that *lock* is a keyword only argument.

   Note that an array of :data:`ctypes.c_char` has *value* and *raw*
   attributes which allow one to use it to store and retrieve strings.
```

The :mod:`multiprocessing.sharedctypes` module

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1490); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1493)

Unknown directive type "module".

```
.. module:: multiprocessing.sharedctypes
   :synopsis: Allocate ctypes objects from shared memory.
```

The :mod:`multiprocessing.sharedctypes` module provides functions for allocating :mod:`ctypes` objects from shared memory which can be inherited by child processes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1496); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1496); *backlink*

Unknown interpreted text role "mod".

Note

Although it is possible to store a pointer in shared memory remember that this will refer to a location in the address space of a specific process. However, the pointer is quite likely to be invalid in the context of a second process and trying to dereference the pointer from the second process may cause a crash.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1508)

Unknown directive type "function".

```
.. function:: RawArray(typecode_or_type, size_or_initializer)

   Return a ctypes array allocated from shared memory.

   *typecode_or_type* determines the type of the elements of the returned array:
   it is either a ctypes type or a one character typecode of the kind used by
   the :mod:`array` module.  If *size_or_initializer* is an integer then it
   determines the length of the array, and the array will be initially zeroed.
   Otherwise *size_or_initializer* is a sequence which is used to initialize the
   array and whose length determines the length of the array.

   Note that setting and getting an element is potentially non-atomic -- use
   :func:`Array` instead to make sure that access is automatically synchronized
   using a lock.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1523)

Unknown directive type "function".

```
.. function:: RawValue(typecode_or_type, *args)

   Return a ctypes object allocated from shared memory.

   *typecode_or_type* determines the type of the returned object: it is either a
   ctypes type or a one character typecode of the kind used by the :mod:`array`
   module.  *\*args* is passed on to the constructor for the type.

   Note that setting and getting the value is potentially non-atomic -- use
   :func:`Value` instead to make sure that access is automatically synchronized
   using a lock.

   Note that an array of :data:`ctypes.c_char` has ``value`` and ``raw``
   attributes which allow one to use it to store and retrieve strings -- see
   documentation for :mod:`ctypes`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1539)

Unknown directive type "function".

```
.. function:: Array(typecode_or_type, size_or_initializer, *, lock=True)

   The same as :func:`RawArray` except that depending on the value of *lock* a
   process-safe synchronization wrapper may be returned instead of a raw ctypes
   array.

   If *lock* is ``True`` (the default) then a new lock object is created to
   synchronize access to the value.  If *lock* is a
   :class:`~multiprocessing.Lock` or :class:`~multiprocessing.RLock` object
   then that will be used to synchronize access to the
   value.  If *lock* is ``False`` then access to the returned object will not be
   automatically protected by a lock, so it will not necessarily be
   "process-safe".

   Note that *lock* is a keyword-only argument.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1555)

Unknown directive type "function".

```
.. function:: Value(typecode_or_type, *args, lock=True)

   The same as :func:`RawValue` except that depending on the value of *lock* a
```

```
process-safe synchronization wrapper may be returned instead of a raw ctypes
object.

If *lock* is ``True`` (the default) then a new lock object is created to
synchronize access to the value.  If *lock* is a :class:`~multiprocessing.Lock` or
:class:`~multiprocessing.RLock` object then that will be used to synchronize access to the
value.  If *lock* is ``False`` then access to the returned object will not be
automatically protected by a lock, so it will not necessarily be
"process-safe".

Note that *lock* is a keyword-only argument.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1570)

Unknown directive type "function".

```
.. function:: copy(obj)

   Return a ctypes object allocated from shared memory which is a copy of the
   ctypes object *obj*.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1575)

Unknown directive type "function".

```
.. function:: synchronized(obj[, lock])

   Return a process-safe wrapper object for a ctypes object which uses *lock* to
   synchronize access.  If *lock* is ``None`` (the default) then a
   :class:`multiprocessing.RLock` object is created automatically.

   A synchronized wrapper will have two methods in addition to those of the
   object it wraps: :meth:`get_obj` returns the wrapped object and
   :meth:`get_lock` returns the lock object used for synchronization.

   Note that accessing the ctypes object through the wrapper can be a lot slower
   than accessing the raw ctypes object.

   .. versionchanged:: 3.5
      Synchronized objects support the :term:`context manager` protocol.
```

The table below compares the syntax for creating shared ctypes objects from shared memory with the normal ctypes syntax. (In the table `MyStruct` is some subclass of :class:`ctypes.Structure`.)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1592); *backlink*

Unknown interpreted text role "class".

ctypes	sharedctypes using type	sharedctypes using typecode
c_double(2.4)	RawValue(c_double, 2.4)	RawValue('d', 2.4)
MyStruct(4, 6)	RawValue(MyStruct, 4, 6)	
(c_short * 7)()	RawArray(c_short, 7)	RawArray('h', 7)
(c_int * 3)(9, 2, 8)	RawArray(c_int, (9, 2, 8))	RawArray('i', (9, 2, 8))

Below is an example where a number of ctypes objects are modified by a child process:

```
from multiprocessing import Process, Lock
from multiprocessing.sharedctypes import Value, Array
from ctypes import Structure, c_double

class Point(Structure):
    _fields_ = [('x', c_double), ('y', c_double)]

def modify(n, x, s, A):
    n.value **= 2
    x.value **= 2
    s.value = s.value.upper()
    for a in A:
        a.x **= 2
        a.y **= 2
```

```
if __name__ == '__main__':
    lock = Lock()

    n = Value('i', 7)
    x = Value(c_double, 1.0/3.0, lock=False)
    s = Array('c', b'hello world', lock=lock)
    A = Array(Point, [(1.875,-6.25), (-5.75,2.0), (2.375,9.5)], lock=lock)

    p = Process(target=modify, args=(n, x, s, A))
    p.start()
    p.join()

    print(n.value)
    print(x.value)
    print(s.value)
    print([(a.x, a.y) for a in A])
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1642)

Unknown directive type "highlight".

```
.. highlight:: none
```

The results printed are

```
49
0.1111111111111111
HELLO WORLD
[(3.515625, 39.0625), (33.0625, 4.0), (5.640625, 90.25)]
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1651)

Unknown directive type "highlight".

```
.. highlight:: python3
```

Managers

Managers provide a way to create data which can be shared between different processes, including sharing over a network between processes running on different machines. A manager object controls a server process which manages *shared objects*. Other processes can access the shared objects by using proxies.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1665)

Unknown directive type "function".

```
.. function:: multiprocessing.Manager()

   Returns a started :class:`~multiprocessing.managers.SyncManager` object which
   can be used for sharing objects between processes.  The returned manager
   object corresponds to a spawned child process and has methods which will
   create shared objects and return corresponding proxies.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1672)

Unknown directive type "module".

```
.. module:: multiprocessing.managers
   :synopsis: Share data between process with shared objects.
```

Manager processes will be shutdown as soon as they are garbage collected or their parent process exits. The manager classes are defined in the :mod:`multiprocessing.managers` module:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1675); *backlink*

Unknown interpreted text role "mod".

Create a BaseManager object.

Once created one should call :meth:`start` or `get_server().serve_forever()` to ensure that the manager object refers to a started manager process.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1683); *backlink*

Unknown interpreted text role "meth".

address is the address on which the manager process listens for new connections. If *address* is `None` then an arbitrary one is chosen.

authkey is the authentication key which will be used to check the validity of incoming connections to the server process. If *authkey* is `None` then `current_process().authkey` is used. Otherwise *authkey* is used and it must be a byte string.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1694)

Unknown directive type "method".

```
.. method:: start([initializer[, initargs]])

   Start a subprocess to start the manager.  If *initializer* is not ``None``
   then the subprocess will call ``initializer(*initargs)`` when it starts.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1699)

Unknown directive type "method".

```
.. method:: get_server()

   Returns a :class:`Server` object which represents the actual server under
   the control of the Manager. The :class:`Server` object supports the
   :meth:`serve_forever` method::

   >>> from multiprocessing.managers import BaseManager
   >>> manager = BaseManager(address=('', 50000), authkey=b'abc')
   >>> server = manager.get_server()
   >>> server.serve_forever()

   :class:`Server` additionally has an :attr:`address` attribute.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1712)

Unknown directive type "method".

```
.. method:: connect()

   Connect a local manager object to a remote manager process::

   >>> from multiprocessing.managers import BaseManager
   >>> m = BaseManager(address=('127.0.0.1', 50000), authkey=b'abc')
   >>> m.connect()
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1720)

Unknown directive type "method".

```
.. method:: shutdown()

   Stop the process used by the manager.  This is only available if
   :meth:`start` has been used to start the server process.

   This can be called multiple times.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1727)

Unknown directive type "method".

```
.. method:: register(typeid[, callable[, proxytype[, exposed[, method_to_typeid[, create_method]]]]
```

```
A classmethod which can be used for registering a type or callable with
the manager class.

*typeid* is a "type identifier" which is used to identify a particular
type of shared object.  This must be a string.

*callable* is a callable used for creating objects for this type
identifier.  If a manager instance will be connected to the
server using the :meth:`connect` method, or if the
*create_method* argument is ``False`` then this can be left as
``None``.

*proxytype* is a subclass of :class:`BaseProxy` which is used to create
proxies for shared objects with this *typeid*.  If ``None`` then a proxy
class is created automatically.

*exposed* is used to specify a sequence of method names which proxies for
this typeid should be allowed to access using
:meth:`BaseProxy._callmethod`.  (If *exposed* is ``None`` then
:attr:`proxytype._exposed_` is used instead if it exists.)  In the case
where no exposed list is specified, all "public methods" of the shared
object will be accessible.  (Here a "public method" means any attribute
which has a :meth:`~object.__call__` method and whose name does not begin
with ``'_'``.)

*method_to_typeid* is a mapping used to specify the return type of those
exposed methods which should return a proxy.  It maps method names to
typeid strings.  (If *method_to_typeid* is ``None`` then
:attr:`proxytype._method_to_typeid_` is used instead if it exists.)  If a
method's name is not a key of this mapping or if the mapping is ``None``
then the object returned by the method will be copied by value.

*create_method* determines whether a method should be created with name
*typeid* which can be used to tell the server process to create a new
shared object and return a proxy for it.  By default it is ``True``.
```

:class:`BaseManager` instances also have one read-only property:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1765); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1767)

Unknown directive type "attribute".

```
.. attribute:: address

   The address used by the manager.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1771)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.3
   Manager objects support the context management protocol -- see
   :ref:`typecontextmanager`.  :meth:`~contextmanager.__enter__` starts the
   server process (if it has not already started) and then returns the
   manager object.  :meth:`~contextmanager.__exit__` calls :meth:`shutdown`.

   In previous versions :meth:`~contextmanager.__enter__` did not start the
   manager's server process if it was not already started.
```

A subclass of :class:`BaseManager` which can be used for the synchronization of processes. Objects of this type are returned by :func:`multiprocessing.Manager`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1782); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1782); *backlink*

Unknown interpreted text role "func".

Its methods create and return :ref:`multiprocessing-proxy_objects` for a number of commonly used data types to be synchronized across processes. This notably includes shared lists and dictionaries.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1786); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1790)

Unknown directive type "method".

```
.. method:: Barrier(parties[, action[, timeout]])

   Create a shared :class:`threading.Barrier` object and return a
   proxy for it.

   .. versionadded:: 3.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1797)

Unknown directive type "method".

```
.. method:: BoundedSemaphore([value])

   Create a shared :class:`threading.BoundedSemaphore` object and return a
   proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1802)

Unknown directive type "method".

```
.. method:: Condition([lock])

   Create a shared :class:`threading.Condition` object and return a proxy for
   it.

   If *lock* is supplied then it should be a proxy for a
   :class:`threading.Lock` or :class:`threading.RLock` object.

   .. versionchanged:: 3.3
      The :meth:`~threading.Condition.wait_for` method was added.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1813)

Unknown directive type "method".

```
.. method:: Event()

   Create a shared :class:`threading.Event` object and return a proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1817)

Unknown directive type "method".

```
.. method:: Lock()

   Create a shared :class:`threading.Lock` object and return a proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1821)

Unknown directive type "method".

```
.. method:: Namespace()

   Create a shared :class:`Namespace` object and return a proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1825)

Unknown directive type "method".

```
.. method:: Queue([maxsize])

   Create a shared :class:`queue.Queue` object and return a proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1829)

Unknown directive type "method".

```
.. method:: RLock()

   Create a shared :class:`threading.RLock` object and return a proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1833)

Unknown directive type "method".

```
.. method:: Semaphore([value])

   Create a shared :class:`threading.Semaphore` object and return a proxy for
   it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1838)

Unknown directive type "method".

```
.. method:: Array(typecode, sequence)

   Create an array and return a proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1842)

Unknown directive type "method".

```
.. method:: Value(typecode, value)

   Create an object with a writable ``value`` attribute and return a proxy
   for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1847)

Unknown directive type "method".

```
.. method:: dict()
            dict(mapping)
            dict(sequence)

   Create a shared :class:`dict` object and return a proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1853)

Unknown directive type "method".

```
   .. method:: list()
               list(sequence)

      Create a shared :class:`list` object and return a proxy for it.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1858)

Unknown directive type "versionchanged".

```
   .. versionchanged:: 3.6
      Shared objects are capable of being nested.  For example, a shared
      container object such as a shared list can contain other shared objects
      which will all be managed and synchronized by the :class:`SyncManager`.
```

A type that can register with :class:`SyncManager`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1865); *backlink*

Unknown interpreted text role "class".

A namespace object has no public methods, but does have writable attributes. Its representation shows the values of its attributes.

However, when using a proxy for a namespace object, an attribute beginning with `'_'` will be an attribute of the proxy and not an attribute of the referent:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1874)

Unknown directive type "doctest".

```
   .. doctest::

    >>> manager = multiprocessing.Manager()
    >>> Global = manager.Namespace()
    >>> Global.x = 10
    >>> Global.y = 'hello'
    >>> Global._z = 12.3    # this is an attribute of the proxy
    >>> print(Global)
    Namespace(x=10, y='hello')
```

Customized managers

To create one's own manager, one creates a subclass of :class:`BaseManager` and uses the :meth:`~BaseManager.register` classmethod to register new types or callables with the manager class. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1888); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1888); *backlink*

Unknown interpreted text role "meth".

```
from multiprocessing.managers import BaseManager

class MathsClass:
    def add(self, x, y):
        return x + y
    def mul(self, x, y):
        return x * y

class MyManager(BaseManager):
    pass

MyManager.register('Maths', MathsClass)

if __name__ == '__main__':
    with MyManager() as manager:
        maths = manager.Maths()
        print(maths.add(4, 3))         # prints 7
```

```
        print(maths.mul(7, 8))          # prints 56
```

Using a remote manager

It is possible to run a manager server on one machine and have clients use it from other machines (assuming that the firewalls involved allow it).

Running the following commands creates a server for a single shared queue which remote clients can access:

```
>>> from multiprocessing.managers import BaseManager
>>> from queue import Queue
>>> queue = Queue()
>>> class QueueManager(BaseManager): pass
>>> QueueManager.register('get_queue', callable=lambda:queue)
>>> m = QueueManager(address=('', 50000), authkey=b'abracadabra')
>>> s = m.get_server()
>>> s.serve_forever()
```

One client can access the server as follows:

```
>>> from multiprocessing.managers import BaseManager
>>> class QueueManager(BaseManager): pass
>>> QueueManager.register('get_queue')
>>> m = QueueManager(address=('foo.bar.org', 50000), authkey=b'abracadabra')
>>> m.connect()
>>> queue = m.get_queue()
>>> queue.put('hello')
```

Another client can also use it:

```
>>> from multiprocessing.managers import BaseManager
>>> class QueueManager(BaseManager): pass
>>> QueueManager.register('get_queue')
>>> m = QueueManager(address=('foo.bar.org', 50000), authkey=b'abracadabra')
>>> m.connect()
>>> queue = m.get_queue()
>>> queue.get()
'hello'
```

Local processes can also access that queue, using the code from above on the client to access it remotely:

```
>>> from multiprocessing import Process, Queue
>>> from multiprocessing.managers import BaseManager
>>> class Worker(Process):
...     def __init__(self, q):
...         self.q = q
...         super().__init__()
...     def run(self):
...         self.q.put('local hello')
...
>>> queue = Queue()
>>> w = Worker(queue)
>>> w.start()
>>> class QueueManager(BaseManager): pass
...
>>> QueueManager.register('get_queue', callable=lambda: queue)
>>> m = QueueManager(address=('', 50000), authkey=b'abracadabra')
>>> s = m.get_server()
>>> s.serve_forever()
```

Proxy Objects

A proxy is an object which *refers* to a shared object which lives (presumably) in a different process. The shared object is said to be the *referent* of the proxy. Multiple proxy objects may have the same referent.

A proxy object has methods which invoke corresponding methods of its referent (although not every method of the referent will necessarily be available through the proxy). In this way, a proxy can be used just like its referent can:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 1986)

Unknown directive type "doctest".

```
.. doctest::

   >>> from multiprocessing import Manager
   >>> manager = Manager()
   >>> l = manager.list([i*i for i in range(10)])
   >>> print(l)
   [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
   >>> print(repr(l))
   <ListProxy object, typeid 'list' at 0x...>
   >>> l[4]
```

```
16
>>> l[2:5]
[4, 9, 16]
```

Notice that applying :func:`str` to a proxy will return the representation of the referent, whereas applying :func:`repr` will return the representation of the proxy.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2000); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2000); *backlink*

Unknown interpreted text role "func".

An important feature of proxy objects is that they are picklable so they can be passed between processes. As such, a referent can contain :ref:`multiprocessing-proxy_objects`. This permits nesting of these managed lists, dicts, and other :ref:`multiprocessing-proxy_objects`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2004); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2004); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2009)

Unknown directive type "doctest".

```
.. doctest::

   >>> a = manager.list()
   >>> b = manager.list()
   >>> a.append(b)         # referent of a now contains referent of b
   >>> print(a, b)
   [<ListProxy object, typeid 'list' at ...>] []
   >>> b.append('hello')
   >>> print(a[0], b)
   ['hello'] ['hello']
```

Similarly, dict and list proxies may be nested inside one another:

```
>>> l_outer = manager.list([ manager.dict() for i in range(2) ])
>>> d_first_inner = l_outer[0]
>>> d_first_inner['a'] = 1
>>> d_first_inner['b'] = 2
>>> l_outer[1]['c'] = 3
>>> l_outer[1]['z'] = 26
>>> print(l_outer[0])
{'a': 1, 'b': 2}
>>> print(l_outer[1])
{'c': 3, 'z': 26}
```

If standard (non-proxy) :class:`list` or :class:`dict` objects are contained in a referent, modifications to those mutable values will not be propagated through the manager because the proxy has no way of knowing when the values contained within are modified. However, storing a value in a container proxy (which triggers a `__setitem__` on the proxy object) does propagate through the manager and so to effectively modify such an item, one could re-assign the modified value to the container proxy:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2033); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2033); *backlink*

Unknown interpreted text role "class".

```
# create a list proxy and append a mutable object (a dictionary)
lproxy = manager.list()
lproxy.append({})
# now mutate the dictionary
d = lproxy[0]
d['a'] = 1
d['b'] = 2
# at this point, the changes to d are not yet synced, but by
# updating the dictionary, the proxy is notified of the change
lproxy[0] = d
```

This approach is perhaps less convenient than employing nested :ref:`multiprocessing-proxy_objects` for most use cases but also demonstrates a level of control over the synchronization.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2052); *backlink*

Unknown interpreted text role "ref".

Note

The proxy types in :mod:`multiprocessing` do nothing to support comparisons by value. So, for instance, we have:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2058); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2061)

Unknown directive type "doctest".

```
.. doctest::

    >>> manager.list([1,2,3]) == [1,2,3]
    False
```

One should just use a copy of the referent instead when making comparisons.

Proxy objects are instances of subclasses of :class:`BaseProxy`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2070); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2072)

Unknown directive type "method".

```
.. method:: _callmethod(methodname[, args[, kwds]])

   Call and return the result of a method of the proxy's referent.

   If ``proxy`` is a proxy whose referent is ``obj`` then the expression ::

      proxy._callmethod(methodname, args, kwds)

   will evaluate the expression ::

      getattr(obj, methodname)(*args, **kwds)

   in the manager's process.

   The returned value will be a copy of the result of the call or a proxy to
   a new shared object -- see documentation for the *method_to_typeid*
   argument of :meth:`BaseManager.register`.
```

```
If an exception is raised by the call, then is re-raised by
:meth:`_callmethod`.  If some other exception is raised in the manager's
process then this is converted into a :exc:`RemoteError` exception and is
raised by :meth:`_callmethod`.

Note in particular that an exception will be raised if *methodname* has
not been *exposed*.

An example of the usage of :meth:`_callmethod`:

.. doctest::

   >>> l = manager.list(range(10))
   >>> l._callmethod('__len__')
   10
   >>> l._callmethod('__getitem__', (slice(2, 7),)) # equivalent to l[2:7]
   [2, 3, 4, 5, 6]
   >>> l._callmethod('__getitem__', (20,))          # equivalent to l[20]
   Traceback (most recent call last):
   ...
   IndexError: list index out of range
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2112)

Unknown directive type "method".

```
.. method:: _getvalue()

   Return a copy of the referent.

   If the referent is unpicklable then this will raise an exception.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2118)

Unknown directive type "method".

```
.. method:: __repr__

   Return a representation of the proxy object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2122)

Unknown directive type "method".

```
.. method:: __str__

   Return the representation of the referent.
```

Cleanup

A proxy object uses a weakref callback so that when it gets garbage collected it deregisters itself from the manager which owns its referent.

A shared object gets deleted from the manager process when there are no longer any proxies referring to it.

Process Pools

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2140)

Unknown directive type "module".

```
.. module:: multiprocessing.pool
   :synopsis: Create pools of processes.
```

One can create a pool of processes which will carry out tasks submitted to it with the :class:`Pool` class.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2143); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2146)

Invalid class attribute value for "class" directive: "Pool([processes[, initializer[, initargs[, maxtasksperchild [, context]]]]])".

```
.. class:: Pool([processes[, initializer[, initargs[, maxtasksperchild [, context]]]]])

   A process pool object which controls a pool of worker processes to which jobs
   can be submitted.  It supports asynchronous results with timeouts and
   callbacks and has a parallel map implementation.

   *processes* is the number of worker processes to use.  If *processes* is
   ``None`` then the number returned by :func:`os.cpu_count` is used.

   If *initializer* is not ``None`` then each worker process will call
   ``initializer(*initargs)`` when it starts.

   *maxtasksperchild* is the number of tasks a worker process can complete
   before it will exit and be replaced with a fresh worker process, to enable
   unused resources to be freed. The default *maxtasksperchild* is ``None``, which
   means worker processes will live as long as the pool.

   *context* can be used to specify the context used for starting
   the worker processes.  Usually a pool is created using the
   function :func:`multiprocessing.Pool` or the :meth:`Pool` method
   of a context object.  In both cases *context* is set
   appropriately.

   Note that the methods of the pool object should only be called by
   the process which created the pool.

   .. warning::
      :class:`multiprocessing.pool` objects have internal resources that need to be
      properly managed (like any other resource) by using the pool as a context manager
      or by calling :meth:`close` and :meth:`terminate` manually. Failure to do this
      can lead to the process hanging on finalization.

      Note that it is **not correct** to rely on the garbage collector to destroy the pool
      as CPython does not assure that the finalizer of the pool will be called
      (see :meth:`object.__del__` for more information).

   .. versionadded:: 3.2
      *maxtasksperchild*

   .. versionadded:: 3.4
      *context*

   .. note::

      Worker processes within a :class:`Pool` typically live for the complete
      duration of the Pool's work queue. A frequent pattern found in other
      systems (such as Apache, mod_wsgi, etc) to free resources held by
      workers is to allow a worker within a pool to complete only a set
      amount of work before being exiting, being cleaned up and a new
      process spawned to replace the old one. The *maxtasksperchild*
      argument to the :class:`Pool` exposes this ability to the end user.

   .. method:: apply(func[, args[, kwds]])

      Call *func* with arguments *args* and keyword arguments *kwds*.  It blocks
      until the result is ready. Given this blocks, :meth:`apply_async` is
      better suited for performing work in parallel. Additionally, *func*
      is only executed in one of the workers of the pool.

   .. method:: apply_async(func[, args[, kwds[, callback[, error_callback]]]])

      A variant of the :meth:`apply` method which returns a
      :class:`~multiprocessing.pool.AsyncResult` object.

      If *callback* is specified then it should be a callable which accepts a
      single argument.  When the result becomes ready *callback* is applied to
      it, that is unless the call failed, in which case the *error_callback*
      is applied instead.

      If *error_callback* is specified then it should be a callable which
      accepts a single argument.  If the target function fails, then
      the *error_callback* is called with the exception instance.

      Callbacks should complete immediately since otherwise the thread which
      handles the results will get blocked.
```

```
.. method:: map(func, iterable[, chunksize])

   A parallel equivalent of the :func:`map` built-in function (it supports only
   one *iterable* argument though, for multiple iterables see :meth:`starmap`).
   It blocks until the result is ready.

   This method chops the iterable into a number of chunks which it submits to
   the process pool as separate tasks.  The (approximate) size of these
   chunks can be specified by setting *chunksize* to a positive integer.

   Note that it may cause high memory usage for very long iterables. Consider
   using :meth:`imap` or :meth:`imap_unordered` with explicit *chunksize*
   option for better efficiency.

.. method:: map_async(func, iterable[, chunksize[, callback[, error_callback]]])

   A variant of the :meth:`.map` method which returns a
   :class:`~multiprocessing.pool.AsyncResult` object.

   If *callback* is specified then it should be a callable which accepts a
   single argument.  When the result becomes ready *callback* is applied to
   it, that is unless the call failed, in which case the *error_callback*
   is applied instead.

   If *error_callback* is specified then it should be a callable which
   accepts a single argument.  If the target function fails, then
   the *error_callback* is called with the exception instance.

   Callbacks should complete immediately since otherwise the thread which
   handles the results will get blocked.

.. method:: imap(func, iterable[, chunksize])

   A lazier version of :meth:`.map`.

   The *chunksize* argument is the same as the one used by the :meth:`.map`
   method.  For very long iterables using a large value for *chunksize* can
   make the job complete **much** faster than using the default value of
   ``1``.

   Also if *chunksize* is ``1`` then the :meth:`!next` method of the iterator
   returned by the :meth:`imap` method has an optional *timeout* parameter:
   ``next(timeout)`` will raise :exc:`multiprocessing.TimeoutError` if the
   result cannot be returned within *timeout* seconds.

.. method:: imap_unordered(func, iterable[, chunksize])

   The same as :meth:`imap` except that the ordering of the results from the
   returned iterator should be considered arbitrary.  (Only when there is
   only one worker process is the order guaranteed to be "correct".)

.. method:: starmap(func, iterable[, chunksize])

   Like :meth:`~multiprocessing.pool.Pool.map` except that the
   elements of the *iterable* are expected to be iterables that are
   unpacked as arguments.

   Hence an *iterable* of ``[(1,2), (3, 4)]`` results in ``[func(1,2),
   func(3,4)]``.

   .. versionadded:: 3.3

.. method:: starmap_async(func, iterable[, chunksize[, callback[, error_callback]]])

   A combination of :meth:`starmap` and :meth:`map_async` that iterates over
   *iterable* of iterables and calls *func* with the iterables unpacked.
   Returns a result object.

   .. versionadded:: 3.3

.. method:: close()

   Prevents any more tasks from being submitted to the pool.  Once all the
   tasks have been completed the worker processes will exit.

.. method:: terminate()

   Stops the worker processes immediately without completing outstanding
   work.  When the pool object is garbage collected :meth:`terminate` will be
   called immediately.

.. method:: join()
```

```
      Wait for the worker processes to exit.  One must call :meth:`close` or
      :meth:`terminate` before using :meth:`join`.

   .. versionadded:: 3.3
      Pool objects now support the context management protocol -- see
      :ref:`typecontextmanager`.  :meth:`~contextmanager.__enter__` returns the
      pool object, and :meth:`~contextmanager.__exit__` calls :meth:`terminate`.
```

The class of the result returned by :meth:`Pool.apply_async` and :meth:`Pool.map_async`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2316); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2316); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2319)

Unknown directive type "method".

```
.. method:: get([timeout])

   Return the result when it arrives.  If *timeout* is not ``None`` and the
   result does not arrive within *timeout* seconds then
   :exc:`multiprocessing.TimeoutError` is raised.  If the remote call raised
   an exception then that exception will be reraised by :meth:`get`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2326)

Unknown directive type "method".

```
.. method:: wait([timeout])

   Wait until the result is available or until *timeout* seconds pass.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2330)

Unknown directive type "method".

```
.. method:: ready()

   Return whether the call has completed.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2334)

Unknown directive type "method".

```
.. method:: successful()

   Return whether the call completed without raising an exception.  Will
   raise :exc:`ValueError` if the result is not ready.

   .. versionchanged:: 3.7
      If the result is not ready, :exc:`ValueError` is raised instead of
      :exc:`AssertionError`.
```

The following example demonstrates the use of a pool:

```
from multiprocessing import Pool
import time

def f(x):
    return x*x
```

```
    if __name__ == '__main__':
        with Pool(processes=4) as pool:         # start 4 worker processes
            result = pool.apply_async(f, (10,)) # evaluate "f(10)" asynchronously in a single process
            print(result.get(timeout=1))        # prints "100" unless your computer is *very* slow

            print(pool.map(f, range(10)))       # prints "[0, 1, 4,..., 81]"

            it = pool.imap(f, range(10))
            print(next(it))                     # prints "0"
            print(next(it))                     # prints "1"
            print(it.next(timeout=1))           # prints "4" unless your computer is *very* slow

            result = pool.apply_async(time.sleep, (10,))
            print(result.get(timeout=1))        # raises multiprocessing.TimeoutError
```

Listeners and Clients

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2372)

Unknown directive type "module".

```
.. module:: multiprocessing.connection
   :synopsis: API for dealing with sockets.
```

Usually message passing between processes is done using queues or by using :class:`~Connection` objects returned by :func:`~multiprocessing.Pipe`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2375); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2375); *backlink*

Unknown interpreted text role "func".

However, the :mod:`multiprocessing.connection` module allows some extra flexibility. It basically gives a high level message oriented API for dealing with sockets or Windows named pipes. It also has support for *digest authentication* using the :mod:`hmac` module, and for polling multiple connections at the same time.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2379); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2379); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2386)

Unknown directive type "function".

```
.. function:: deliver_challenge(connection, authkey)

   Send a randomly generated message to the other end of the connection and wait
   for a reply.

   If the reply matches the digest of the message using *authkey* as the key
   then a welcome message is sent to the other end of the connection.  Otherwise
   :exc:`~multiprocessing.AuthenticationError` is raised.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2395)

Unknown directive type "function".

```
.. function:: answer_challenge(connection, authkey)

   Receive a message, calculate the digest of the message using *authkey* as the
   key, and then send the digest back.

   If a welcome message is not received, then
   :exc:`~multiprocessing.AuthenticationError` is raised.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2403)

Unknown directive type "function".

```
.. function:: Client(address[, family[, authkey]])

   Attempt to set up a connection to the listener which is using address
   *address*, returning a :class:`~Connection`.

   The type of the connection is determined by *family* argument, but this can
   generally be omitted since it can usually be inferred from the format of
   *address*. (See :ref:`multiprocessing-address-formats`)

   If *authkey* is given and not None, it should be a byte string and will be
   used as the secret key for an HMAC-based authentication challenge. No
   authentication is done if *authkey* is None.
   :exc:`~multiprocessing.AuthenticationError` is raised if authentication fails.
   See :ref:`multiprocessing-auth-keys`.
```

A wrapper for a bound socket or Windows named pipe which is 'listening' for connections.

address is the address to be used by the bound socket or named pipe of the listener object.

> **Note**
>
> If an address of '0.0.0.0' is used, the address will not be a connectable end point on Windows. If you require a connectable end-point, you should use '127.0.0.1'.

family is the type of socket (or named pipe) to use. This can be one of the strings `'AF_INET'` (for a TCP socket), `'AF_UNIX'` (for a Unix domain socket) or `'AF_PIPE'` (for a Windows named pipe). Of these only the first is guaranteed to be available. If *family* is `None` then the family is inferred from the format of *address*. If *address* is also `None` then a default is chosen. This default is the family which is assumed to be the fastest available. See :ref:`multiprocessing-address-formats`. Note that if *family* is `'AF_UNIX'` and address is `None` then the socket will be created in a private temporary directory created using :func:`tempfile.mkstemp`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2432); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2432); *backlink*

Unknown interpreted text role "func".

If the listener object uses a socket then *backlog* (1 by default) is passed to the :meth:`~socket.socket.listen` method of the socket once it has been bound.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2443); *backlink*

Unknown interpreted text role "meth".

If *authkey* is given and not None, it should be a byte string and will be used as the secret key for an HMAC-based authentication challenge. No authentication is done if *authkey* is None. :exc:`~multiprocessing.AuthenticationError` is raised if authentication fails. See :ref:`multiprocessing-auth-keys`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2447); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2447); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2453)

Unknown directive type "method".

```
.. method:: accept()

   Accept a connection on the bound socket or named pipe of the listener
   object and return a :class:`~Connection` object.
   If authentication is attempted and fails, then
   :exc:`~multiprocessing.AuthenticationError` is raised.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2460)

Unknown directive type "method".

```
.. method:: close()

   Close the bound socket or named pipe of the listener object.  This is
   called automatically when the listener is garbage collected.  However it
   is advisable to call it explicitly.
```

Listener objects have the following read-only properties:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2468)

Unknown directive type "attribute".

```
.. attribute:: address

   The address which is being used by the Listener object.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2472)

Unknown directive type "attribute".

```
.. attribute:: last_accepted

   The address from which the last accepted connection came.  If this is
   unavailable then it is ``None``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2477)

Unknown directive type "versionadded".

```
.. versionadded:: 3.3
   Listener objects now support the context management protocol -- see
   :ref:`typecontextmanager`.  :meth:`~contextmanager.__enter__` returns the
   listener object, and :meth:`~contextmanager.__exit__` calls :meth:`close`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2482)

Unknown directive type "function".

```
.. function:: wait(object_list, timeout=None)

   Wait till an object in *object_list* is ready.  Returns the list of
   those objects in *object_list* which are ready.  If *timeout* is a
   float then the call blocks for at most that many seconds.  If
   *timeout* is ``None`` then it will block for an unlimited period.
   A negative timeout is equivalent to a zero timeout.

   For both Unix and Windows, an object can appear in *object_list* if
   it is
```

```
      * a readable :class:`~multiprocessing.connection.Connection` object;
      * a connected and readable :class:`socket.socket` object; or
      * the :attr:`~multiprocessing.Process.sentinel` attribute of a
        :class:`~multiprocessing.Process` object.

      A connection or socket object is ready when there is data available
      to be read from it, or the other end has been closed.

      **Unix**: ``wait(object_list, timeout)`` almost equivalent
      ``select.select(object_list, [], [], timeout)``.  The difference is
      that, if :func:`select.select` is interrupted by a signal, it can
      raise :exc:`OSError` with an error number of ``EINTR``, whereas
      :func:`wait` will not.

      **Windows**: An item in *object_list* must either be an integer
      handle which is waitable (according to the definition used by the
      documentation of the Win32 function ``WaitForMultipleObjects()``)
      or it can be an object with a :meth:`fileno` method which returns a
      socket handle or pipe handle.  (Note that pipe handles and socket
      handles are **not** waitable handles.)

      .. versionadded:: 3.3
```

Examples

The following server code creates a listener which uses `'secret password'` as an authentication key. It then waits for a connection and sends some data to the client:

```
from multiprocessing.connection import Listener
from array import array

address = ('localhost', 6000)     # family is deduced to be 'AF_INET'

with Listener(address, authkey=b'secret password') as listener:
    with listener.accept() as conn:
        print('connection accepted from', listener.last_accepted)

        conn.send([2.25, None, 'junk', float])

        conn.send_bytes(b'hello')

        conn.send_bytes(array('i', [42, 1729]))
```

The following code connects to the server and receives some data from the server:

```
from multiprocessing.connection import Client
from array import array

address = ('localhost', 6000)

with Client(address, authkey=b'secret password') as conn:
    print(conn.recv())                  # => [2.25, None, 'junk', float]

    print(conn.recv_bytes())            # => 'hello'

    arr = array('i', [0, 0, 0, 0, 0])
    print(conn.recv_bytes_into(arr))    # => 8
    print(arr)                          # => array('i', [42, 1729, 0, 0, 0])
```

The following code uses :func:`~multiprocessing.connection.wait` to wait for messages from multiple processes at once:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2555); *backlink*

Unknown interpreted text role "func".

```
import time, random
from multiprocessing import Process, Pipe, current_process
from multiprocessing.connection import wait

def foo(w):
    for i in range(10):
        w.send((i, current_process().name))
    w.close()

if __name__ == '__main__':
    readers = []

    for i in range(4):
        r, w = Pipe(duplex=False)
```

```
            readers.append(r)
            p = Process(target=foo, args=(w,))
            p.start()
            # We close the writable end of the pipe now to be sure that
            # p is the only process which owns a handle for it.  This
            # ensures that when p closes its handle for the writable end,
            # wait() will promptly report the readable end as being ready.
            w.close()

        while readers:
            for r in wait(readers):
                try:
                    msg = r.recv()
                except EOFError:
                    readers.remove(r)
                else:
                    print(msg)
```

Address Formats

- An `'AF_INET'` address is a tuple of the form `(hostname, port)` where *hostname* is a string and *port* is an integer.
- An `'AF_UNIX'` address is a string representing a filename on the filesystem.
- An `'AF_PIPE'` address is a string of the form :samp:`r'\\\\.\\pipe\\{PipeName}'`. To use :func:`Client` to connect to a named pipe on a remote computer called *ServerName* one should use an address of the form :samp:`r'\\\\{ServerName}\\pipe\\{PipeName}'` instead.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2602); *backlink*
>
> Unknown interpreted text role "samp".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2602); *backlink*
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2602); *backlink*
>
> Unknown interpreted text role "samp".

Note that any string beginning with two backslashes is assumed by default to be an `'AF_PIPE'` address rather than an `'AF_UNIX'` address.

Authentication keys

When one uses :meth:`Connection.recv <Connection.recv>`, the data received is automatically unpickled. Unfortunately unpickling data from an untrusted source is a security risk. Therefore :class:`Listener` and :func:`Client` use the :mod:`hmac` module to provide digest authentication.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2616); *backlink*
>
> Unknown interpreted text role "meth".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2616); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2616); *backlink*
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2616); *backlink*

Unknown interpreted text role "mod".

An authentication key is a byte string which can be thought of as a password: once a connection is established both ends will demand proof that the other knows the authentication key. (Demonstrating that both ends are using the same key does **not** involve sending the key over the connection.)

If authentication is requested but no authentication key is specified then the return value of `current_process().authkey` is used (see :class:`~multiprocessing.Process`). This value will be automatically inherited by any :class:`~multiprocessing.Process` object that the current process creates. This means that (by default) all processes of a multi-process program will share a single authentication key which can be used when setting up connections between themselves.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2628); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2628); *backlink*

Unknown interpreted text role "class".

Suitable authentication keys can also be generated by using :func:`os.urandom`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2636); *backlink*

Unknown interpreted text role "func".

Logging

Some support for logging is available. Note, however, that the :mod:`logging` package does not use process shared locks so it is possible (depending on the handler type) for messages from different processes to get mixed up.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2642); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2646)

Unknown directive type "currentmodule".

```
.. currentmodule:: multiprocessing
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2647)

Unknown directive type "function".

```
.. function:: get_logger()

   Returns the logger used by :mod:`multiprocessing`.  If necessary, a new one
   will be created.

   When first created the logger has level :data:`logging.NOTSET` and no
   default handler. Messages sent to this logger will not by default propagate
   to the root logger.

   Note that on Windows child processes will only inherit the level of the
   parent process's logger -- any other customization of the logger will not be
   inherited.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2660)

Unknown directive type "currentmodule".

```
.. currentmodule:: multiprocessing
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2661)

Unknown directive type "function".

```
.. function:: log_to_stderr(level=None)

   This function performs a call to :func:`get_logger` but in addition to
   returning the logger created by get_logger, it adds a handler which sends
   output to :data:`sys.stderr` using format
   ``'[%(levelname)s/%(processName)s] %(message)s'``.
   You can modify ``levelname`` of the logger by passing a ``level`` argument.
```

Below is an example session with logging turned on:

```
>>> import multiprocessing, logging
>>> logger = multiprocessing.log_to_stderr()
>>> logger.setLevel(logging.INFO)
>>> logger.warning('doomed')
[WARNING/MainProcess] doomed
>>> m = multiprocessing.Manager()
[INFO/SyncManager-...] child process calling self.run()
[INFO/SyncManager-...] created temp directory /.../pymp-...
[INFO/SyncManager-...] manager serving at '/.../listener-...'
>>> del m
[INFO/MainProcess] sending shutdown message to manager
[INFO/SyncManager-...] manager exiting with exitcode 0
```

For a full table of logging levels, see the :mod:`logging` module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2684); *backlink*

Unknown interpreted text role "mod".

The :mod:`multiprocessing.dummy` module

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2687); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2690)

Unknown directive type "module".

```
.. module:: multiprocessing.dummy
   :synopsis: Dumb wrapper around threading.
```

:mod:`multiprocessing.dummy` replicates the API of :mod:`multiprocessing` but is no more than a wrapper around the :mod:`threading` module.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2693); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2693); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2693); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2696)

Unknown directive type "currentmodule".

```
.. currentmodule:: multiprocessing.pool
```

In particular, the `Pool` function provided by :mod:`multiprocessing.dummy` returns an instance of :class:`ThreadPool`, which is a subclass of :class:`Pool` that supports all the same method calls but uses a pool of worker threads rather than worker processes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2698); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2698); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2698); *backlink*

Unknown interpreted text role "class".

A thread pool object which controls a pool of worker threads to which jobs can be submitted. :class:`ThreadPool` instances are fully interface compatible with :class:`Pool` instances, and their resources must also be properly managed, either by using the pool as a context manager or by calling :meth:`~multiprocessing.pool.Pool.close` and :meth:`~multiprocessing.pool.Pool.terminate` manually.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2706); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2706); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2706); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2706); *backlink*

Unknown interpreted text role "meth".

processes is the number of worker threads to use. If *processes* is `None` then the number returned by :func:`os.cpu_count` is used.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2713); *backlink*

Unknown interpreted text role "func".

If *initializer* is not `None` then each worker process will call `initializer(*initargs)` when it starts.

Unlike :class:`Pool`, *maxtasksperchild* and *context* cannot be provided.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2719); *backlink*

Unknown interpreted text role "class".

Note

A :class:`ThreadPool` shares the same interface as :class:`Pool`, which is designed around a pool of processes and predates the introduction of the :class:`concurrent.futures` module. As such, it inherits some operations that don't make sense for a pool backed by threads, and it has its own type for representing the status of asynchronous jobs, :class:`AsyncResult`, that is not understood by any other libraries.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 2723); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 2723); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 2723); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 2723); *backlink*

Unknown interpreted text role "class".

Users should generally prefer to use :class:`concurrent.futures.ThreadPoolExecutor`, which has a simpler interface that was designed around threads from the start, and which returns :class:`concurrent.futures.Future` instances that are compatible with many other libraries, including :mod:`asyncio`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 2730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 2730); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 2730); *backlink*

Unknown interpreted text role "mod".

Programming guidelines

There are certain guidelines and idioms which should be adhered to when using :mod:`multiprocessing`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main) (Doc) (library)multiprocessing.rst, line 2742); *backlink*

Unknown interpreted text role "mod".

All start methods

The following applies to all start methods.

Avoid shared state

As far as possible one should try to avoid shifting large amounts of data between processes.

It is probably best to stick to using queues or pipes for communication between processes rather than using the lower level synchronization primitives.

Picklability

Ensure that the arguments to the methods of proxies are picklable.

Thread safety of proxies

Do not use a proxy object from more than one thread unless you protect it with a lock.

(There is never a problem with different processes using the *same* proxy.)

Joining zombie processes

On Unix when a process finishes but has not been joined it becomes a zombie. There should never be very many because each time a new process starts (or :func:`~multiprocessing.active_children` is called) all completed processes which have not yet been joined will be joined. Also calling a finished process's :meth:`Process.is_alive <multiprocessing.Process.is_alive>` will join the process. Even so it is probably good practice to explicitly join all the processes that you start.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2773); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2773); *backlink*

Unknown interpreted text role "meth".

Better to inherit than pickle/unpickle

When using the *spawn* or *forkserver* start methods many types from :mod:`multiprocessing` need to be picklable so that child processes can use them. However, one should generally avoid sending shared objects to other processes using pipes or queues. Instead you should arrange the program so that a process which needs access to a shared resource created elsewhere can inherit it from an ancestor process.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2783); *backlink*

Unknown interpreted text role "mod".

Avoid terminating processes

Using the :meth:`Process.terminate <multiprocessing.Process.terminate>` method to stop a process is liable to cause any shared resources (such as locks, semaphores, pipes and queues) currently being used by the process to become broken or unavailable to other processes.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2793); *backlink*

Unknown interpreted text role "meth".

Therefore it is probably best to only consider using :meth:`Process.terminate <multiprocessing.Process.terminate>` on processes which never use any shared resources.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2799); *backlink*

Unknown interpreted text role "meth".

Joining processes that use queues

Bear in mind that a process that has put items in a queue will wait before terminating until all the buffered items are fed by the "feeder" thread to the underlying pipe. (The child process can call the :meth:`Queue.cancel_join_thread <multiprocessing.Queue.cancel_join_thread>` method of the queue to avoid this behaviour.)

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2805); *backlink*
>
> Unknown interpreted text role "meth".

This means that whenever you use a queue you need to make sure that all items which have been put on the queue will eventually be removed before the process is joined. Otherwise you cannot be sure that processes which have put items on the queue will terminate. Remember also that non-daemonic processes will be joined automatically.

An example which will deadlock is the following:

```
from multiprocessing import Process, Queue

def f(q):
    q.put('X' * 1000000)

if __name__ == '__main__':
    queue = Queue()
    p = Process(target=f, args=(queue,))
    p.start()
    p.join()                    # this deadlocks
    obj = queue.get()
```

A fix here would be to swap the last two lines (or simply remove the `p.join()` line).

Explicitly pass resources to child processes

On Unix using the *fork* start method, a child process can make use of a shared resource created in a parent process using a global resource. However, it is better to pass the object as an argument to the constructor for the child process.

Apart from making the code (potentially) compatible with Windows and the other start methods this also ensures that as long as the child process is still alive the object will not be garbage collected in the parent process. This might be important if some resource is freed when the object is garbage collected in the parent process.

So for instance

```
from multiprocessing import Process, Lock

def f():
    ... do something using "lock" ...

if __name__ == '__main__':
    lock = Lock()
    for i in range(10):
        Process(target=f).start()
```

should be rewritten as

```
from multiprocessing import Process, Lock

def f(l):
    ... do something using "l" ...

if __name__ == '__main__':
    lock = Lock()
    for i in range(10):
        Process(target=f, args=(lock,)).start()
```

Beware of replacing :data:`sys.stdin` with a "file like object"

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2872); *backlink*
>
> Unknown interpreted text role "data".

:mod:`multiprocessing` originally unconditionally called:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2874); *backlink*
>
> Unknown interpreted text role "mod".

```
os.close(sys.stdin.fileno())
```

in the :meth:`multiprocessing.Process._bootstrap` method --- this resulted in issues with processes-in-processes. This has been changed to:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2878); *backlink*

Unknown interpreted text role "meth".

```
sys.stdin.close()
sys.stdin = open(os.open(os.devnull, os.O_RDONLY), closefd=False)
```

Which solves the fundamental issue of processes colliding with each other resulting in a bad file descriptor error, but introduces a potential danger to applications which replace :func:`sys.stdin` with a "file-like object" with output buffering. This danger is that if multiple processes call :meth:`~io.IOBase.close()` on this file-like object, it could result in the same data being flushed to the object multiple times, resulting in corruption.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2884); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2884); *backlink*

Unknown interpreted text role "meth".

If you write a file-like object and implement your own caching, you can make it fork-safe by storing the pid whenever you append to the cache, and discarding the cache when the pid changes. For example:

```
@property
def cache(self):
    pid = os.getpid()
    if pid != self._pid:
        self._pid = pid
        self._cache = []
    return self._cache
```

For more information, see :issue:`5155`, :issue:`5313` and :issue:`5331`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2903); *backlink*

Unknown interpreted text role "issue".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2903); *backlink*

Unknown interpreted text role "issue".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2903); *backlink*

Unknown interpreted text role "issue".

The *spawn* and *forkserver* start methods

There are a few extra restriction which don't apply to the *fork* start method.

More picklability

Ensure that all arguments to :meth:`Process.__init__` are picklable. Also, if you subclass :class:`~multiprocessing.Process` then make sure that instances will be picklable when the :meth:`Process.start <multiprocessing.Process.start>` method is called.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2913); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2913); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2913); *backlink*

Unknown interpreted text role "meth".

Global variables

Bear in mind that if code run in a child process tries to access a global variable, then the value it sees (if any) may not be the same as the value in the parent process at the time that :meth:`Process.start <multiprocessing.Process.start>` was called.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2920); *backlink*

Unknown interpreted text role "meth".

However, global variables which are just module level constants cause no problems.

Safe importing of main module

Make sure that the main module can be safely imported by a new Python interpreter without causing unintended side effects (such a starting a new process).

For example, using the *spawn* or *forkserver* start method running the following module would fail with a :exc:`RuntimeError`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2934); *backlink*

Unknown interpreted text role "exc".

```
from multiprocessing import Process

def foo():
    print('hello')

p = Process(target=foo)
p.start()
```

Instead one should protect the "entry point" of the program by using `if __name__ == '__main__':` as follows:

```
from multiprocessing import Process, freeze_support, set_start_method

def foo():
    print('hello')

if __name__ == '__main__':
    freeze_support()
    set_start_method('spawn')
    p = Process(target=foo)
    p.start()
```

(The `freeze_support()` line can be omitted if the program will be run normally instead of frozen.)

This allows the newly spawned Python interpreter to safely import the module and then run the module's `foo()` function.

Similar restrictions apply if a pool or manager is created in the main module.

Examples

Demonstration of how to create and use customized managers and proxies:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2977)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/mp_newtype.py
   :language: python3
```

Using :class:`~multiprocessing.pool.Pool`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2981); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2983)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/mp_pool.py
   :language: python3
```

An example showing how to use queues to feed tasks to a collection of worker processes and collect the results:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\(cpython-main)(Doc)(library)multiprocessing.rst, line 2990)

Unknown directive type "literalinclude".

```
.. literalinclude:: ../includes/mp_workers.py
```