Python's multiprocessing.shared_memory

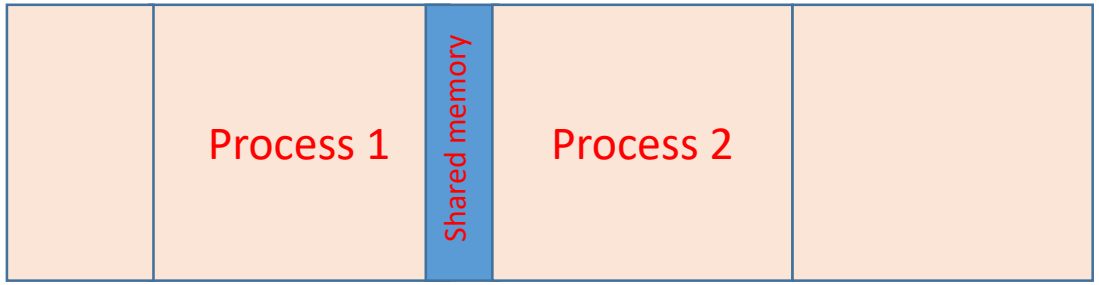
CPEN333 - 2021 W2

University of British Columbia

©Farshid Agharebparast

Introduction

- One mechanism for inter-process communication is via shared memory.
  - Recall that normally, the OS strictly protects memory allocated to each process from other processes.
  - But by setting up shared memory, two processes can exchange data through that shared memory.



- In this set of slides, we examine the `multiprocessing.shared_memory` class that would allows us to set up and control shared memory.

Objectives

- To use Python's `shared_memory` to implement shared-memory-based IPC

- To introduce `ShareabeList` and `SharedMemoryManger`

SharedMemory class

- The `multiprocessing.shared_memory` module includes the class `SharedMemory`. It can be used to allocate and manage shared memory for access by two or more processes.

- `class multiprocessing.shared_memory.SharedMemory(name=None, create=False, size=0)`
  - Creates a new shared memory block or attaches to an existing shared memory block.
  - https://docs.python.org/3/library/multiprocessing.shared_memory.html#multiprocessing.shared_memory.SharedMemory

SharedMemory class

- https://docs.python.org/3/library/multiprocessing.shared_memory.html#multiprocessing.shared_memory.SharedMemory

class `multiprocessing.shared_memory.`**SharedMemory**(*name=None*, *create=False*, *size=0*)

Creates a new shared memory block or attaches to an existing shared memory block. Each shared memory block is assigned a unique name. In this way, one process can create a shared memory block with a particular name and a different process can attach to that same shared memory block using that same name.

As a resource for sharing data across processes, shared memory blocks may outlive the original process that created them. When one process no longer needs access to a shared memory block that might still be needed by other processes, the `close()` method should be called. When a shared memory block is no longer needed by any process, the `unlink()` method should be called to ensure proper cleanup.

name is the unique name for the requested shared memory, specified as a string. When creating a new shared memory block, if `None` (the default) is supplied for the name, a novel name will be generated.

create controls whether a new shared memory block is created (`True`) or an existing shared memory block is attached (`False`).

size specifies the requested number of bytes when creating a new shared memory block. Because some platforms choose to allocate chunks of memory based upon that platform's memory page size, the exact size of the shared memory block may be larger or equal to the size requested. When attaching to an existing shared memory block, the `size` parameter is ignored.

SharedMemory (cont.)

- The initializer's arguments are as follows:
  - `name` is a string that specifies a unique name for the requested shared memory.
  - `create` is a Boolean that specifies creating a new shard memory (`True`) or attaching an already existent one (`False`).
  - `size` specifies the size of the shared memory block in bytes.
- There are also the following methods and properties:
  - `close()` : closes access to the shared memory for this instance
  - `Unlink()`: requests the destruction of the underlying shared memory
  - `buf`: memory-view of the shared memory content
  - `name`: read-only access to *name*
  - `size`: read-only access to *size*

When to use `close` or `unlink`

From the documentation:

- `close`: When one process no longer needs access to a shared memory block that might still be needed by other processes, the `close`() method should be called.
  - does not cause the shared memory block itself to be destroyed
  - "to ensure proper cleanup of resources, all instances should call close() once the instance is no longer needed"

  https://docs.python.org/3/library/multiprocessing.shared_memory.html#multiprocessing.shared_memory.SharedMemory.close
- `unlink`: When a shared memory block is no longer needed by any process, the `unlink`() method should be called to ensure proper cleanup.
  - "should be called once (and only once) across all processes which have need for the shared memory block"

  https://docs.python.org/3/library/multiprocessing.shared_memory.html#multiprocessing.shared_memory.SharedMemory.unlink

Example

```
from multiprocessing import shared_memory, Process
import numpy as np

def process2(dataSize, dataType):
    existing_shm = shared_memory.SharedMemory(name="demoSM")
    b = np.ndarray((dataSize,), dtype=dataType, buffer=existing_shm.buf)
    print(f"Child process has access to {b}")
    existing_shm.close() #clean up from within the child process

if __name__ == "__main__":
    a = np.array([1, 1, 2, 3, 5, 8])
    shm = shared_memory.SharedMemory(name="demoSM", create=True, size=a.nbytes)
    b = np.ndarray(a.shape, dtype=a.dtype, buffer=shm.buf)
    b[:]=a[:]
    p = Process(target=process2, kwargs={"dataSize": a.size, "dataType": type(a[0])})
    p.start()
    p.join()
    shm.close() #clean up from within the main process
    shm.unlink() #free and release the shared memory block at the very end
```

Attaching to the existing shared memory block

Creating a shared memory block

Output:

Child process has access to [1 1 2 3 5 8]

Note: numpy is used here as numpy arrays are convenient to be used for this case.

Modified version of an example in https://docs.python.org/3/library/multiprocessing.shared_memory.html#module-multiprocessing.shared_memory

SharedMemoryManager

- https://docs.python.org/3/library/multiprocessing.shared_memory.html#multiprocessing.managers.SharedMemoryManager

class `multiprocessing.managers.` **SharedMemoryManager**([*address*[, *authkey*]])

A subclass of `BaseManager` which can be used for the management of shared memory blocks across processes.

A call to `start()` on a `SharedMemoryManager` instance causes a new process to be started. This new process's sole purpose is to manage the life cycle of all shared memory blocks created through it. To trigger the release of all shared memory blocks managed by that process, call `shutdown()` on the instance. This triggers a `SharedMemory.unlink()` call on all of the `SharedMemory` objects managed by that process and then stops the process itself. By creating `SharedMemory` instances through a `SharedMemoryManager`, we avoid the need to manually track and trigger the freeing of shared memory resources.

This class provides methods for creating and returning `SharedMemory` instances and for creating a list-like object (`ShareableList`) backed by shared memory.

Refer to `multiprocessing.managers.BaseManager` for a description of the inherited *address* and *authkey* optional input arguments and how they may be used to connect to an existing `SharedMemoryManager` service from other processes.

SharedMemory(*size*)

Create and return a new `SharedMemory` object with the specified `size` in bytes.

ShareableList(*sequence*)

Create and return a new `ShareableList` object, initialized by the values from the input `sequence`.

ShareableList

- https://docs.python.org/3/library/multiprocessing.shared_memory.html#multiprocessing.shared_memory.ShareableList

class `multiprocessing.shared_memory.` **`ShareableList`**(*sequence=None*, *, *name=None*)

Provides a mutable list-like object where all values stored within are stored in a shared memory block. This constrains storable values to only the `int`, `float`, `bool`, `str` (less than 10M bytes each), `bytes` (less than 10M bytes each), and `None` built-in data types. It also notably differs from the built-in `list` type in that these lists can not change their overall length (i.e. no append, insert, etc.) and do not support the dynamic creation of new `ShareableList` instances via slicing.

sequence is used in populating a new `ShareableList` full of values. Set to `None` to instead attach to an already existing `ShareableList` by its unique shared memory name.

name is the unique name for the requested shared memory, as described in the definition for `SharedMemory`. When attaching to an existing `ShareableList`, specify its shared memory block's unique name while leaving `sequence` set to `None`.

References

- Python documentation
  - https://docs.python.org/3/library/multiprocessing.shared_memory.html