<u>Instructional Team</u> Office Hours <u>Progress</u> <u>Course</u> **Discussion** <u>Dates</u> (1) ☆ Course / Lesson 7: Concurrency & Async Programming / Lesson 7 Content Previous Next > **Part 8: Multiprocessing** ☐ Bookmark this page

## Part 8: Multiprocessing

## Threading and multiprocessing

```
    processes are completely isolated
    no locking :) (and no GIL!)
    instead of locking: messaging
```

multiprocessing provides an API very similar to threading, so the transition is easy

use multiprocessing. Process instead of threading. Thread

```
import multiprocessing
import os
import time

def func():
    print "hello from process %s" % os.getpid()
    time.sleep(1)

proc = multiprocessing.Process(target=func, args=())
proc.start()
proc = multiprocessing.Process(target=func, args=())
proc.start()
```

# **Differences with Threading**

Multiprocessing has its own multiprocessing. Queue which handles interprocess communication.

Also has its own versions of Lock, RLock, Semaphore

```
from multiprocessing import Queue, Lock
```

multiprocessing. Pipe for 2-way process communication:

```
from multiprocessing import Pipe
parent_conn, child_conn = Pipe()
child_conn.send("foo")
print parent_conn.recv()
```

## Messaging

#### Pipes (multiprocessing.Pipe)

- Returns a pair of connected objects.
- Largely mimics Unix pipes, but higher level.
- send pickled objects or buffers.

#### Queues (multiprocessing.Queue)

- same interface as queue.Queue
- implemented on top of nines

	_		Next > t threaded programs using queues to multiprocessing queue is and unpickled to pass between processes – significant overhead.	- 1	
Othe	er features o	of the multiprocessing	package	© All Rig	ghts Reserved



© 2024 University of Washington | Seattle, WA. All rights reserved.

Help Center Contact Us Privacy Terms

Built on Open  $\operatorname{col}$  by RACCOONGANG  $\Leftrightarrow$ 

 ${\tt edX}, {\tt Open\,edX\,and\,the\,edX\,and\,Open\,edX\,logos\,are\,trademarks\,or\,registered\,trademarks\,of\,edX\,lnc.}$