Part 8: Multiprocessing

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 pipes

- implemented on top of pipes.

‹ Previous Next ›

- means you can pretty easily port threaded programs using queues to multiprocessing. - queue is the only shared data. - data is all pickled and unpickled to pass between processes – significant overhead.

Other features of the multiprocessing package