Python Threading and Synchronization

CS 360 Internet Programming

Daniel Zappala

Brigham Young University
Computer Science Department

Threading Module

- high-level interface to threads
- includes
  - threads
  - timers
  - mutexes and condition variables
  - semaphores
  - events

Thread Objects

1. create a subclass of Thread
2. override the run() method
3. create an instance of the object
4. call the start() method on the instance to start the thread's run method

- all other methods on the thread object can be called, but they will run on the object, not on the thread
- see http://docs.python.org/library/threading.html

Thread Object

```
1  class MyThread(threading.Thread):
2      def __init__(self):
3          threading.Thread.__init__(self)
4          threading.Thread.daemon = True
5      def run(self):
6          while True:
7              # do work
```

- must call the Thread superclass `__init__()` method
- the daemon flag causes the thread to terminate if the main thread exits
- alternatively, the parent thread can call `join()` on the thread object and wait for it to exit

Thread Example

- *see example code on web site*

Timers

- run a method at some time in the future
- uses a separate thread

```
1  def hello():
2      print ''Hello World''
3
4  t = Timer(10, hello)
5  t.start()
```

Mutex/Lock Objects

```
1  lock = Threading.Lock()
2  lock.acquire()
3  # critical section
4  lock.release()
```

Condition Objects

- automatically creates an associated mutex

```
1  cv = threading.Condition()
```

```
1  # Producer
2  cv.acquire()
3  makeitem()
4  cv.notify()
5  cv.release()
```

```
1  # Consumer
2  cv.acquire()
3  while not available():
4      cv.wait()
5  getitem()
6  cv.release()
```

Semaphores

```
1 sem = threading.Semaphore()
2 spaces = threading.Semaphore(100)
3 slots = threading.Semaphore(0)
```

```
1 # Producer
2 spaces.acquire()
3 sem.acquire()
4 makeitem()
5 sem.release()
6 slots.release()
```

```
1 # Consumer
2 slots.acquire()
3 sem.acquire()
4 getitem()
5 sem.release()
6 spaces.release()
```

Shared Memory

- all the previous synchronization examples have assumed that the synchronization variables are stored in shared memory
- can store them in shared memory
- better way is to pass shared memory to each thread object
- *see example code on web site*