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](http://docs.python.org/library/threading.html)
Thread Object

```python
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

```python
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

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

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

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

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

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

# Consumer
1 slots.acquire()
2 sem.acquire()
3 getitem()
4 sem.release()
5 spaces.release()
```
all the previous synchronization examples have assumed that the synchronization variables are stored in shared memory.

can store them in shared memory.

good way is to pass shared memory to each thread object.

see example code on web site.