# BAB I

# PYTHON MULTITHREADED PROGRAMMING

Menjalankan beberapa *thread* mirip dengan menjalankan beberapa program yang berbeda secara bersamaan, namun dengan manfaat berikut :

- Beberapa *thread* dalam proses berbagi ruang data yang sama dengan benang induk dan karena dapat saling berbagi informasi atau berkomunikasi satu sama lain dengan lebih muda daripada jika prosesnya terpisah
- *thread* terkadang disebut proses ringan dan tidak membutuhkan banyak memori atas, mereka lebih murah daripada proses.

Sebuah *thread* memiliki permulaan, urutan eksekusi dan sebuah kesimpulan. Ini memiliki pointer perintah yang melacak dari mana dalam konteksnya saat ini berjalan.

- Hal ini dapat dilakukan sebelum pre-*empted* (*inturrepted*)
- Untuk sementara dapat ditunda sementara *thread* lainnya yang sedang berjalan ini disebut unggul.

## 1.1 Memulai *Thread* Baru

Untuk melakukan *thread* lain, perlu memanggil metode berikut yang tersedia dimodul *thread* :

Thread.start _new _thread (function, args [, kwargs] )

Pemanggilan metode ini memungkinkan cara cepat dan tepat untuk membuat *thread* baru di linux dan window.

Pemanggilan metode segera kembali dan anak *thread* dimulai dan fungsi pemanggilan dengan daftar *args* telah berlalu. Saat fungsi kembali ujung *thread* akan berakhir.

Disini, *args* adalah tupel argumen. Gunakan tupel kosong untuk memanggil fungsi tanpa melewati argumen. *Kwargs* adalah kamus opsional argumen kata kunci.

Contoh :

```
#!/usr/bin/python

Import thread
Import time

# Define a function for the thread
Def print _time (threadNamw, delay):
        Count = 0
        While count <5:
```

```
        Time.sleep(delay)
        Count +=1
        Print ” %s : %s ” % (threadName, time.ctime(time.time()))

# Create two thread as follows
try:
thread.start _new _thread(print _time, ( ”Thread-1 ”, 2, ))
thread.start _new _thread(print _time, ( ”Thread-2 ”, 4,))
except:
  print ”Error: unable to start thread ”

while 1:
pass
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut :

```
Thread-1 : Thu Jan 22 15:42:17 2009

Thread-1 : Thu Jan 22 15:42:19 2009

Thread-2 : Thu Jan 22 15:42:19 2009

Thread-1 : Thu Jan 22 15:42:21 2009

Thread-2 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:25 2009

Thread-2 : Thu Jan 22 15:42:27 2009

Thread-2 : Thu Jan 22 15:42:31 2009

Thread-2 : Thu Jan 22 15:42:35 2009
```

Meskipun sangat efektif untuk benang tingkat rendah, namun modul *thread* sangat terbatas dibandingkan dengan modul yang baru.

### 1.2 Modul Threading

Modul threading yang lebih baru disertakan dengan Python 2.4 memberikan jauh lebih kuat, dukungan tingkat tinggi untuk *thread* dari modul *thread* dibahas pada bagian sebelumnya.

The *threading* modul mengekpos semua metode dari *thread* dan menyediakan beberapa metode tambahan :

- **threading.activeCount()**

  Mengembalikan jumlah objek *thread* yang aktif

- **threading.currentThread()**

  Mengembalikan jumlah objek *thread* dalam kontrol benang pemanggil

- **threading.enumerate()**

  Mengembalikan daftar semua benda *thread* yang sedang aktif

  Selain metode, modul *threading* memiliki *thread* kelas yang mengimplementasikan *threading.* Metode yang disediakan oleh *thread* kelas adalah sebagai berikut :

- **run()**

  Metode adalah titik masuk untuk *thread*

- **start()**

  Metode dimulai *thread* dengan memanggil metode run

- **join([time])**

  Menunggu benang untuk mengakhiri

- **isAlive()**

  Metode memeriksa apakah *thread* masih mengeksekusi

- **getName()**

  Metode mengambalikan nama *thread*

- **setName()**

  Metode menetapkan nama *thread*

### 1.3 Membuat *Thread* Menggunakan *Threading* Modul

Untuk melaksanakan *thread* baru menggunakan *threading* harus melakukan hal berikut :

- **M**endefinisikan subclass dari *thread* kelas
- Menimpa _init _ (self [args]) metode untuk menambahkan argumen tambahan
- Menimpa run(self[args]) metode untuk menerapkan apa *thread* harus dilakukan ketika mulai

Setelah membuat baru *thread* subclass, dapat membuah seuah instance dari itu dan kemudian memulai *thread* baru dengan menerapkan *start(),* yang ada gilirinnya panggilan *run()* metode.

Contoh :

```
#!/usr/bin/python

import threading
import time

exitFlag = 0

class myThread (threading.Thread):
        def _init _(self, threadID, name, counter) :
    threading.Thread. _init _(self)
        self.threadID = threadID
        self.name = name
```

```
        self.counter = counter
def run (self) :
                print ”Starting ” + self.name
                print _time(self.name, self.counter, 5)
                print ”Exiting ”+ self.name

def print _time(threadName, delay, counter):
while counter:
                if exitFlag:
                        threadName.exit()
                time.sleep(delay)
                print ” %s: %s ” % (threadName, time.ctime(time.time()))
counter -= 1

# Create new threads
thread1 = myThread(1, ”Thread-1 ”, 1)
thread2 = myThread(2, ”Thread-2 ”, 2)

# Start new threads
thread1.start()
thread2.start()
print ”Exiting Main Thread ”
```

Ketika kode diatas dijalankan, menghasilkan hasil sebagai berikut:

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1 : Thu Mar 21 09:10:03 2013
Thread-1 : Thu Mar 21 09:10:04 2013
Thread-2 : Thu Mar 21 09:10:04 2013
Thread-1 : Thu Mar 21 09:10:05 2013
Thread-2 : Thu Mar 21 09:10:06 2013
Thread-1 : Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2 : Thu Mar 21 09:10:08 2013
Thread-2 : Thu Mar 21 09:10:10 2013
Thread-2 : Thu Mar 21 09:10:12 2013
Exiting Thread=2
```

### 1.4 Sinkronisasi *Thread*

*Threading* modul disediakan dengan Python termasuk sederhana untuk menerapkan mekanisme bahwa memungkinkan untuk menyinkronkan *thread* penguncian. Sebuah kunci baru dibuat dengan memanggil *lock()* metode yang mengembalikan kunci baru.

The *acquire (blocking)* metode objek kunci baru digunakan untuk memaksa *thread* untuk menjalankan serempak. Opsional *blocking* parameter memungkikan untuk mengontrol apakah *thread* menunggu untuk mendapatkan kunci.

Jika *blocking* diatur ke 0, *thread* segera kembali dengan nilai 0 jika kunci tidak dapat diperoleh dan dengan 1 jika kunci dikuisisi. Jika pemblokiran diatur ke 1, blok dan menunggu kunci yang akan dirilis.

The *release()* metode objek kunci baru digunakan untuk melepaskan kunci ketika tidak lagi diperlukan.

### 1.5 Multithreaded Antrian Prioritas

The queue modul memungkinkan untuk membuat objek antrian baru yang dapat menampung jumlah tertentu item. Ada metode berikut untuk mengontrol antrian :

- **get()**

  Menghapus dan mengembalikan item dari antrian

- **put()**

  Menambahkan item ke antrian

- **qsize()**

  Mengembalikan jumlah item yang saat ini dalam antrian

- **empty()**

  Mengembalikan benar jika antrian kosong jika tidak, salah

- **full()**

  Mengembalikan benar jika antrian penuh jika tidak, salah