Multiprozessing

Jan Popko

Python Advanced

Business Trends Academy (BTA) GmbH

bta
BUSINESS TRENDS ACADEMY

Nestorstraße 36
D-10709 Berlin

Tel.: +49 (0) 30 894 087 57
Fax: +49 (0) 30 895 429 94

Geschäftsführer:
Gabriele Fleischmann-Hahn
Maxi-Marlen Fleischmann
Hauptsitz des Unternehmens:
Nestorstraße 36, D-10709 Berlin
HRB 115251 B / Amtsgericht Berlin Charlottenburg
Steuer-Nr. 27/248/31179

Multiprozessing & Multithreading

- beides Wege für Multitasking

Multiprozessing:

- Programm läuft parallel in multiplen Prozessen
- Jeder Prozess hat eigenen Address Space
- Umgeht GIL (Global Interpreter Lock)
- Fehler oder Memory Leak beeinflußen andere Prozesse nicht
- Taxiert den Speicher mehr
- Heavy weight

Multithreading:

- Programm läuft parallel in einem Prozess
- Teilen sich den Address Space
- Light weight

Klasse Pool

- Pool Object, kontrolliert einen Pool von Worker Prozessen
- Verteilt Prozesse auf die CPUs
- Argumente:
    - processes: Maximale Nummer von gleichzeitigen Prozessen
        - Standard = Anzahl der CPUs
    - initializer(*initargs): wenn nicht None, starten alle Prozesse mit den initializer
    - maxtasksperchild: Nummer der Tasks jeder Worker Prozess erledigt
        - Standard = Tasks werden bearbeitet, bis der Pool leer ist

```
import multiprocessing

# Pool Objekt erstellen
pool = multiprocessing.Pool()

# Liste auf eine Funktion mappen
result = pool.map(func, iterable)
```

Klasse Process

```
# Prozess Erstellen mit der Klasse Process

from multiprocessing import Process

def func(x):
        #does something

if __name__ == '__main__':
        # Erstellt neuen Prozess
        newprocess = Process(target=func,args=(x,))

        # Startet den neuen Prozess
        newprocess.start()

        # Programm wartet darauf, daß der Prozess beendet wird
        newprocess.join()
```

Methoden für Prozesse

```
# stoppt den Prozess (ohne ihn abzuschließen!)
Process.terminate()

# gibt den gerade ablaufenden Prozess wieder
multiprocessing.current_process()

# gibt den Namen des Prozesses wieder
multiprocessing.current_process().name

# gibt die Prozess ID wieder
multiprocessing.current_process().pid
```


Business Trends Academy (BTA) GmbH
bta BUSINESS TRENDS ACADEMY
Nestorstraße 36
D-10709 Berlin
Tel.: +49 (0) 30 894 087 57
Fax: +49 (0) 30 895 429 94
Geschäftsführer:
Gabriele Fleischmann-Hahn
Maxi-Marlen Fleischmann
Hauptsitz des Unternehmens:
Nestorstraße 36, D-10709 Berlin
HRB 115251 B / Amtsgericht Berlin Charlottenburg
Steuer-Nr. 27/248/31179

Shared Memory Datentypen

class Value:

- Returns ctype Objekt, welches dem geteilten Speicher zugeteilt wurde
- Value(type,args)
  - type: ist ein ctype (Integer, Boolean, Double, Char)
  - args: wird auf den Datentyp angewand
- Objekt selbst kann mit .value aufgerufen werden

class Array:

- Returns ctype Array, welches dem geteilten Speicher zugeteilt wurde
- Array(type,size_or_initializer)
  - type: ist ein ctype (Integer, Boolean, Double, Char)
    - Array beinhaltet nur Daten dieses Typs
  - size_or_initializer: gibt die direkte Größe des Arrays an
    - oder Initialisierer (range(x))

Klasse Lock

- Benutzt zum Synchronisieren von Prozessen
- Wenn das Schloß (Lock) für Objekt aktiviert ist kann kein anderer Prozess darauf zugreifen

```
# Modul importieren
from multiprocessing import Lock
def func(x,lock):
        # Schloß wird gesetzt
        lock.acquire()
        # do something with x
        # Schloß wird freigegeben
        lock.release()
if __name__ == '__main__':
# Lock() Objekt erstellen
        lock = Lock()
process = Process(target=func, args=(x, lock))
```

Klasse multiprocessing.Queue

Erstellt First In First Out Schlangenobjekt

```
import multiprocessing

q = multiprocessing.Queue(maxsize = 0)
# wenn maxsize > 0 →anzahl der möglichen Elemente in einer Queue
# sonst unendlich
```

.put(item) – hängt Element an die Schlange, wenn sie nicht voll ist
.get(item) – gibt Element aus der Schlange und entfernt es daraus

- Die folgenden Methoden sind keine Garantie dafür, daß .put() und .get() funktionieren:
.qsize() - gibt die geschätzte länge der Schlange wieder
.empty() - True wenn die Schlange leer ist, sonst False
.full() - True wenn die Schlange voll ist, sonst False

Klasse multiprocessing.Pipe

Pipe(duplex = True): ein beidseitiges Verbindungstupel
Pipe(False): das erste element kann nur receiven und das zweite nur senden

```
import multiprocessing

def f(conn):
    conn.send([42, None, 'Hello World!'])
    conn.close()

if __name__ == '__main__':
    parent_conn, child_conn = Pipe()
    p = Process(target=f, args=(child_conn,))
    p.start()
    print(parent_conn.recv())
      # gibt: "[42, None, 'Hello World!']"
    print(parent_conn.poll()
      # gibt an ob elemente in der Pipe sind
    p.join()
```