

## Modul 7

### Python Pemrograman Jaringan Multithreading

Pemrograman multithreading adalah teknik yang memungkinkan eksekusi simultan dari beberapa bagian (thread) dalam sebuah program. Ini sangat berguna dalam pemrograman modern untuk memanfaatkan prosesor multi-core dan meningkatkan performa aplikasi. Mari kita lihat lebih dalam tentang konsep ini, serta bagaimana mengimplementasikannya menggunakan Python.

#### 1. Konsep Dasar Multithreading

**Thread** adalah unit terkecil dari eksekusi yang dapat dilakukan secara independen oleh proses. Dalam pemrograman multithreading, beberapa thread dapat dijalankan bersamaan dalam satu proses, berbagi sumber daya yang sama seperti memori.

##### Manfaat Multithreading:

1. **Responsivitas:** Aplikasi tetap responsif karena satu thread tidak menghalangi eksekusi thread lain. Contoh: Antarmuka pengguna (UI) yang tidak membeku saat melakukan proses latar belakang.
2. **Efisiensi:** Memanfaatkan CPU multi-core dengan membagi pekerjaan di antara thread, meningkatkan performa aplikasi.
3. **Pemrograman Paralel:** Memungkinkan beberapa bagian dari program berjalan bersamaan, seperti dalam pemrosesan data besar atau simulasi kompleks.

#### 2. Implementasi Multithreading di Python

Python menyediakan modul `threading` untuk mendukung pemrograman multithreading. Berikut adalah contoh dasar menggunakan `threading`.

##### Penjelasan:

- `target` menentukan fungsi yang akan dijalankan dalam thread.
- `start()` memulai eksekusi thread.
- `join()` menunggu hingga thread selesai.

##### Penggunaan Threading dalam Aplikasi Nyata

1. **Multithreading untuk IO-bound Operations:** Ideal untuk aplikasi yang banyak melakukan operasi input/output, seperti membaca file, atau komunikasi jaringan. Misalnya, aplikasi web yang melayani beberapa permintaan HTTP secara bersamaan.
2. **Multithreading untuk CPU-bound Operations:** Terbatas dalam Python karena Global Interpreter Lock (GIL), yang mencegah beberapa thread Python untuk menjalankan kode

bytecode secara bersamaan. Namun, bisa menggunakan modul `multiprocessing` untuk benar-benar memanfaatkan CPU multi-core.

### Contoh Dasar Multithreading:

```
import threading
import time

# Fungsi yang akan dijalankan dalam thread
def print_numbers():
    for i in range(5):
        print(f"Number: {i}")
        time.sleep(1)

def print_letters():
    for letter in 'ABCDE':
        print(f"Letter: {letter}")
        time.sleep(1.5)

# Membuat thread
thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)

# Memulai thread
thread1.start()
thread2.start()

# Menunggu thread selesai
thread1.join()
thread2.join()

print("Selesai.")
```

Ln: 3 Col: 0

```
IDLE Shell 3.12.4
Python 3.12.4 (v3.12.4:8e8a4baf65, Jun 6 2024, 17:33:18) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/user/Desktop/program modul 7/multithreading.py =====
Number: 0Letter: A

Number: 1
Letter: B
Number: 2
Letter: CNumber: 3

Number: 4
Letter: D
Letter: E
Selesai.
>>>
```

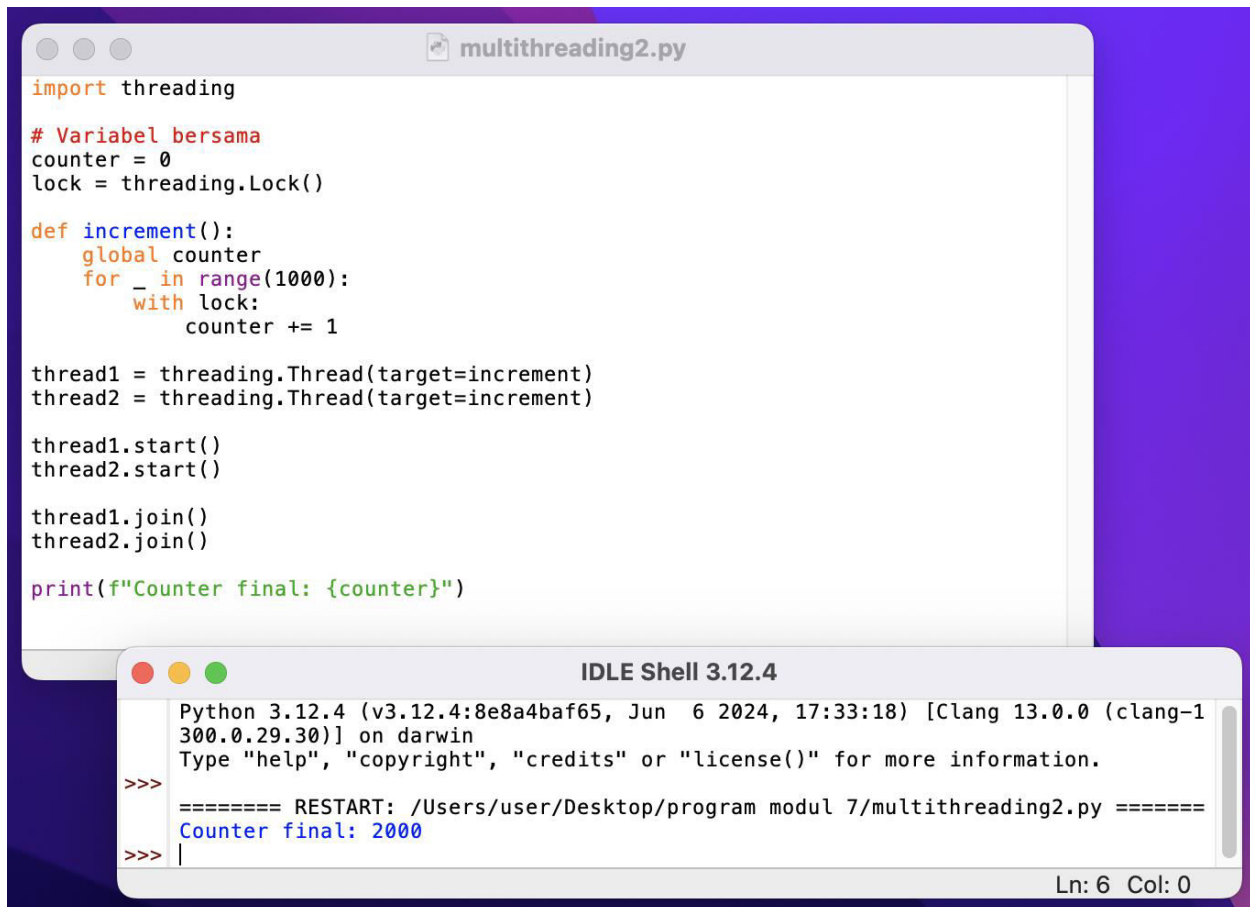
Ln: 8 Col: 0

### 3. Pemrograman Multithreading Lanjutan

#### Sinkronisasi Thread

Ketika beberapa thread mengakses data bersama, perlu memastikan data tetap konsisten menggunakan mekanisme sinkronisasi seperti `Lock`.

#### Contoh Sinkronisasi dengan Lock:



The screenshot shows a Python IDE window titled 'multithreading2.py' and an 'IDLE Shell 3.12.4' window. The code in the IDE defines a shared counter and a lock, then creates two threads that increment the counter. The shell output shows the final counter value is 2000.

```
import threading

# Variabel bersama
counter = 0
lock = threading.Lock()

def increment():
    global counter
    for _ in range(1000):
        with lock:
            counter += 1

thread1 = threading.Thread(target=increment)
thread2 = threading.Thread(target=increment)

thread1.start()
thread2.start()

thread1.join()
thread2.join()

print(f"Counter final: {counter}")
```

```
Python 3.12.4 (v3.12.4:8e8a4baf65, Jun 6 2024, 17:33:18) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/user/Desktop/program modul 7/multithreading2.py =====
Counter final: 2000
>>> |
```

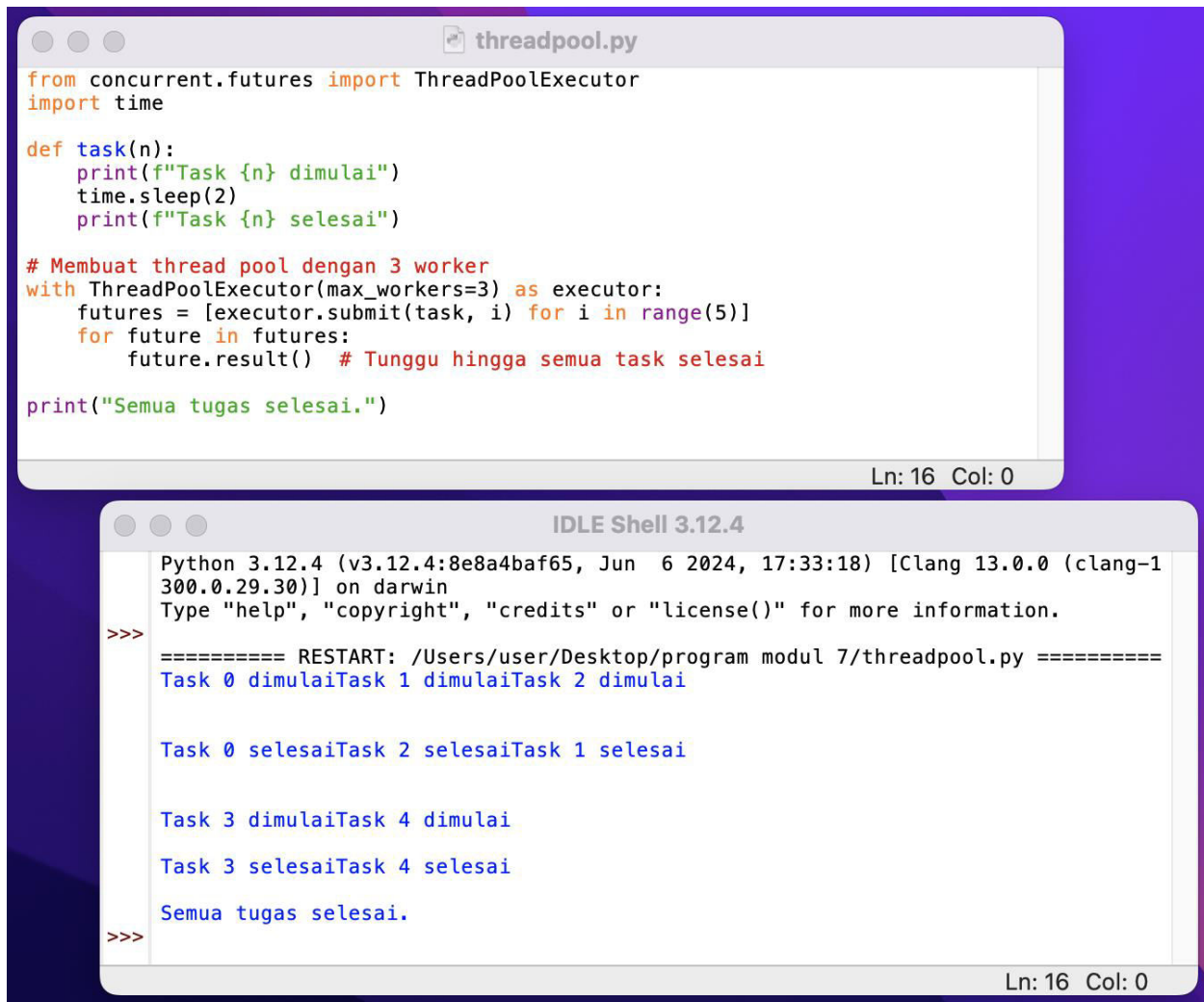
Ln: 6 Col: 0

#### Penjelasan:

- `Lock` digunakan untuk memastikan hanya satu thread yang mengakses `counter` pada satu waktu, menghindari kondisi balapan (race condition).

#### Thread Pools

Untuk manajemen thread yang lebih efisien, gunakan `ThreadPoolExecutor` dari modul `concurrent.futures`.



The image shows a screenshot of a code editor with two windows. The top window, titled 'threadpool.py', contains a Python script that uses the `ThreadPoolExecutor` from the `concurrent.futures` module to execute five tasks in parallel. Each task prints its start and end times and sleeps for 2 seconds. The bottom window, titled 'IDLE Shell 3.12.4', shows the output of running the script. The output displays the start and end of five tasks, with tasks 0, 1, and 2 completing first, followed by tasks 3 and 4, and finally the completion of all tasks.

```
threadpool.py
from concurrent.futures import ThreadPoolExecutor
import time

def task(n):
    print(f"Task {n} dimulai")
    time.sleep(2)
    print(f"Task {n} selesai")

# Membuat thread pool dengan 3 worker
with ThreadPoolExecutor(max_workers=3) as executor:
    futures = [executor.submit(task, i) for i in range(5)]
    for future in futures:
        future.result() # Tunggu hingga semua task selesai

print("Semua tugas selesai.")

Ln: 16 Col: 0
```

```
IDLE Shell 3.12.4
Python 3.12.4 (v3.12.4:8e8a4baf65, Jun 6 2024, 17:33:18) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/user/Desktop/program modul 7/threadpool.py =====
Task 0 dimulaiTask 1 dimulaiTask 2 dimulai

Task 0 selesaiTask 2 selesaiTask 1 selesai

Task 3 dimulaiTask 4 dimulai
Task 3 selesaiTask 4 selesai
Semua tugas selesai.
>>>

Ln: 16 Col: 0
```

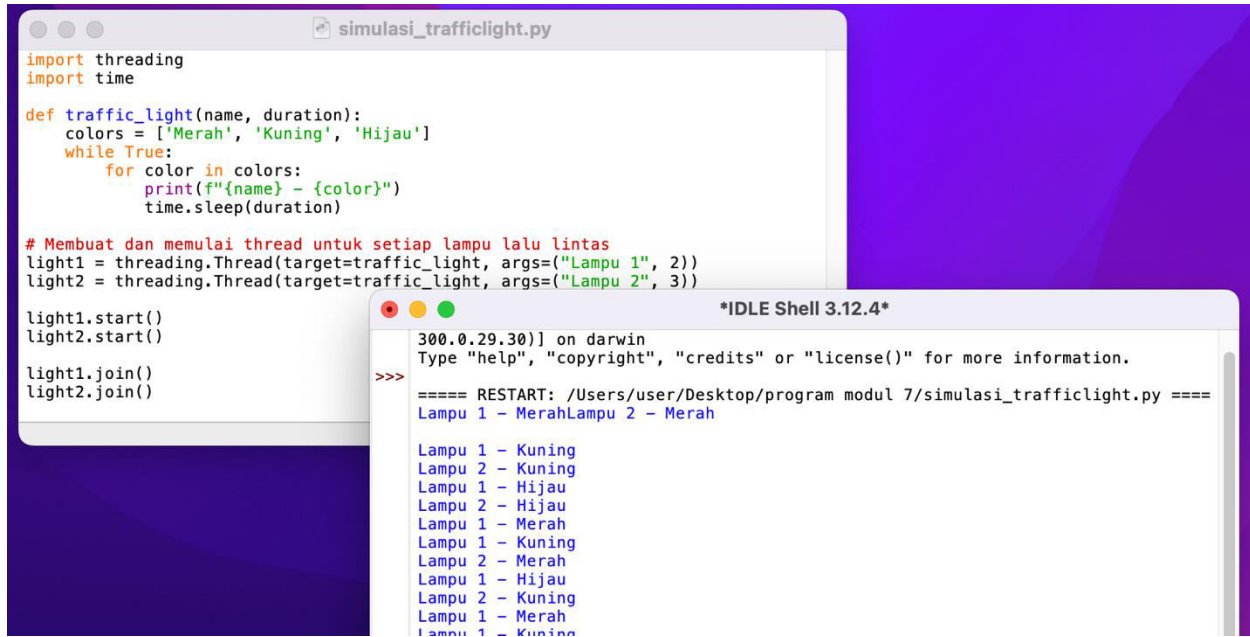
## Penjelasan:

- `ThreadPoolExecutor` mengelola thread pool dan mengatur jumlah maksimum thread yang aktif.
- `submit()` menambahkan tugas ke thread pool

### 3. Simulasi Traffic Light

Simulasi traffic light adalah contoh sederhana di mana beberapa thread mewakili lampu lalu lintas yang berganti warna.

#### Contoh: Simulasi Traffic Light



The image shows a screenshot of a Python IDE with two windows. The top window, titled 'simulasi\_trafficlight.py', contains the following Python code:

```
import threading
import time

def traffic_light(name, duration):
    colors = ['Merah', 'Kuning', 'Hijau']
    while True:
        for color in colors:
            print(f"{name} - {color}")
            time.sleep(duration)

# Membuat dan memulai thread untuk setiap lampu lalu lintas
light1 = threading.Thread(target=traffic_light, args=("Lampu 1", 2))
light2 = threading.Thread(target=traffic_light, args=("Lampu 2", 3))

light1.start()
light2.start()

light1.join()
light2.join()
```

The bottom window, titled '\*IDLE Shell 3.12.4\*', shows the output of the script:

```
300.0.29.30]] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/user/Desktop/program modul 7/simulasi_trafficlight.py =====
Lampu 1 - MerahLampu 2 - Merah
Lampu 1 - Kuning
Lampu 2 - Kuning
Lampu 1 - Hijau
Lampu 2 - Hijau
Lampu 1 - Merah
Lampu 1 - Kuning
Lampu 2 - Merah
Lampu 1 - Hijau
Lampu 2 - Kuning
Lampu 1 - Merah
Lampu 1 - Kuning
```

#### Penjelasan:

- Setiap lampu lalu lintas diwakili oleh thread yang secara bergantian menampilkan warna.
- `time.sleep` digunakan untuk mengatur durasi setiap warna.

## 4. Kesalahan Umum dan Tips

- **Deadlock:** Terjadi ketika dua atau lebih thread saling menunggu satu sama lain untuk melepaskan resource yang diperlukan.
  - **Solusi:** Gunakan timeout pada `Lock`, atau pertimbangkan desain ulang untuk menghindari ketergantungan silang.
- **Race Condition:** Terjadi ketika dua atau lebih thread mengakses data bersama tanpa sinkronisasi, menghasilkan hasil yang tidak terduga.
  - **Solusi:** Gunakan mekanisme sinkronisasi seperti `Lock` untuk melindungi data bersama.
- **GIL:** Global Interpreter Lock membatasi eksekusi kode bytecode Python secara bersamaan.
  - **Solusi:** Untuk tugas CPU-bound, pertimbangkan menggunakan `multiprocessing` alih-alih `threading`.

Pemrograman multithreading memungkinkan efisiensi dan responsivitas aplikasi dengan menjalankan beberapa thread secara bersamaan. Teknik ini sangat berguna untuk aplikasi yang memerlukan eksekusi paralel, seperti server web, aplikasi IO-bound, dan sistem yang memerlukan sinkronisasi data. Memahami dan mengelola thread dengan benar adalah kunci untuk menghindari masalah seperti deadlock dan race condition, serta memanfaatkan keunggulan pemrograman paralel.