Tugas Pemograman Paralel

Disusun Oleh :

Agil Deriansyah Hasan
4522210125

Dosen Pengampu :

Febri Maspiyanti, S.Kom., M.Kom
Pemograman Paralel - A

S1-Teknik Informatika
Fakultas Teknik
Universitas Pancasila 2023/2024

Eksperimen 1 : Menambah Thread Tambahan

```
import threading
import time

# Fungsi untuk menampilkan angka
def print_numbers():
    for i in range(1, 6):
        time.sleep(1)
        print(f"Number: {i}")

# Fungsi untuk menampilkan huruf
def print_letters():
    for letter in ['A', 'B', 'C', 'D', 'E']:
        time.sleep(1.5)
        print(f"Letter: {letter}")

# Fungsi untuk menampilkan simbol
def print_symbols():
    for symbol in ['!', '@', '#', '$', '%']:
        time.sleep(0.7)
        print(f"Symbol: {symbol}")

# Membuat tiga thread
thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)
thread3 = threading.Thread(target=print_symbols)

# Memulai thread
print("Starting threads...")
thread1.start()
thread2.start()
thread3.start()

# Menunggu semua thread selesai
thread1.join()
thread2.join()
thread3.join()

print("All threads have finished execution.")\
```

Hasil Running Eksperimen 1

```
Starting threads...
Symbol: !
Number: 1
Symbol: @
Letter: A
Number: 2
Symbol: #
Symbol: $
Letter: B
Number: 3
Symbol: %
Number: 4
Letter: C
Number: 5
Letter: D
Letter: E
All threads have finished execution.
```

Eksperimen 2 : Mengatur Prioritas dengan Thread.daemon

```python
import threading
import time
# Fungsi untuk menampilkan angka
def print_numbers():
    for i in range(1, 6):
        time.sleep(1)
        print(f"Number: {i}")

# Fungsi untuk menampilkan huruf
def print_letters():
    for letter in ['A', 'B', 'C', 'D', 'E']:
        time.sleep(1.5)
        print(f"Letter: {letter}")

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

# Menjadikan thread kedua sebagai daemon
thread2.daemon = True

# Memulai thread
print("Starting threads...")
thread1.start()
thread2.start()

# Menunggu thread pertama selesai (thread daemon tidak dijamin selesai)
thread1.join()
print("Main thread has finished execution.")
```

Hasil Running Eksperimen 2

```
Starting threads...
Number: 1
Letter: A
Number: 2
Number: 3
Letter: B
Number: 4
Letter: C
Number: 5
Main thread has finished execution.
Letter: D
Letter: E
```

Eksperimen 3 : Sinkronisasi dengan lock

```python
import threading
import time

# Membuat lock
lock = threading.Lock()

# Fungsi untuk menampilkan angka
def print_numbers():
    for i in range(1, 6):
        time.sleep(1)
        # Mengunci sebelum mencetak
        lock.acquire()
        print(f"Number: {i}")
        lock.release()

# Fungsi untuk menampilkan huruf
def print_letters():
    for letter in ['A', 'B', 'C', 'D', 'E']:
        time.sleep(1.5)
        # Mengunci sebelum mencetak
        lock.acquire()
        print(f"Letter: {letter}")
        lock.release()
# Membuat dua thread
thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)
# Memulai thread
print("Starting threads...")
thread1.start()
thread2.start()
# Menunggu thread selesai
thread1.join()
thread2.join()
print("Both threads have finished execution.")
```

Hasil Running Eksperimen 3

```
Starting threads...
Number: 1
Letter: A
Number: 2
Letter: B
Number: 3
Number: 4
Letter: C
Number: 5
Letter: D
Letter: E
Both threads have finished execution.
```

Eksperimen 4 : Menangkap Exception dari Thread

```python
import threading
import time

# Fungsi untuk menampilkan angka
def print_numbers():
    try:
        for i in range(1, 6):
            if i == 3:
                raise ValueError("Error on number 3!")
            time.sleep(1)
            print(f"Number: {i}")
    except Exception as e:
        print(f"Exception caught in print_numbers: {e}")

# Fungsi untuk menampilkan huruf
def print_letters():
    try:
        for letter in ['A', 'B', 'C', 'D', 'E']:
            time.sleep(1.5)
            print(f"Letter: {letter}")
    except Exception as e:
        print(f"Exception caught in print_letters: {e}")

# Membuat dua thread
thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)
# Memulai thread
print("Starting threads...")
thread1.start()
thread2.start()
# Menunggu thread selesai
thread1.join()
thread2.join()
print("Both threads have finished execution.")
```

Hasil Running Eksperimen 4

```
Starting threads...
Number: 1
Letter: A
Number: 2
Exception caught in print_numbers: Error on number 3!
Letter: B
Letter: C
Letter: D
Letter: E
Both threads have finished execution.
```

Eksperimen 5 : Thread dengan Argumen

```python
import threading
import time

# Fungsi dengan argumen
def print_custom_range(start, end):
    for i in range(start, end + 1):
        time.sleep(1)
        print(f"Number: {i}")

# Membuat thread dengan argumen
thread1 = threading.Thread(target=print_custom_range, args=(1, 5))
thread2 = threading.Thread(target=print_custom_range, args=(6, 10))

# Memulai thread
print("Starting threads...")
thread1.start()
thread2.start()

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

print("Both threads have finished execution.")
```

Hasil Running Eksperimen 5

```
Starting threads...
Number: 1Number: 6

Number: 2Number: 7

Number: 3Number: 8

Number: 9Number: 4

Number: 10Number: 5

Both threads have finished execution.
```

Eksperimen 6 : Menggunakan Queue untuk Komunikasi Antar Thread

```python
import threading
import time
import queue

# Fungsi untuk menambahkan angka ke dalam queue
def add_to_queue(q):
    for i in range(1, 6):
        time.sleep(1)
        q.put(i)
        print(f"Added {i} to queue")

# Fungsi untuk mengambil angka dari queue
def consume_from_queue(q):
    while not q.empty() or thread1.is_alive():  # Menunggu hingga thread1 selesai
        try:
            item = q.get(timeout=1)
            print(f"Consumed {item} from queue")
        except queue.Empty:
            pass

# Membuat queue
q = queue.Queue()

# Membuat thread
thread1 = threading.Thread(target=add_to_queue, args=(q,))
thread2 = threading.Thread(target=consume_from_queue, args=(q,))

# Memulai thread
print("Starting threads...")
thread1.start()
thread2.start()
# Menunggu thread selesai
thread1.join()
thread2.join()
print("Both threads have finished execution.")
```

Hasil Running Eksperimen 6

```
Starting threads...
Added 1 to queueConsumed 1 from queue

Added 2 to queueConsumed 2 from queue

Added 3 to queueConsumed 3 from queue

Added 4 to queueConsumed 4 from queue

Added 5 to queueConsumed 5 from queue

Both threads have finished execution.
```

Eksperimen 7 : Menghentikan Thread dengan Event

```python
import threading
import time

# Membuat event
stop_event = threading.Event()

# Fungsi untuk mencetak angka, berhenti jika event ter-set
def print_numbers():
    for i in range(1, 11):
        if stop_event.is_set():
            print("Stopping thread early!")
            break
        time.sleep(1)
        print(f"Number: {i}")

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

# Memulai thread
print("Starting thread...")
thread1.start()

# Menghentikan thread setelah 5 detik
time.sleep(5)
stop_event.set()

# Menunggu thread selesai
thread1.join()
print("Thread stopped.")
```

Hasil Running Eksperimen 7

```
Starting thread...
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
Stopping thread early!
Thread stopped.
```

Eksperimen 8 : Menggunakan ThreadPoolExecutor untuk Pooling Thread

```python
import time
from concurrent.futures import ThreadPoolExecutor

# Fungsi untuk mencetak angka dengan delay
def print_number(n):
    time.sleep(1)
    print(f"Number: {n}")

# Membuat thread pool dengan maksimal 3 worker
with ThreadPoolExecutor(max_workers=3) as executor:
    for i in range(1, 11):
        executor.submit(print_number, i)

print("All tasks have been submitted.")
```

Hasil Running Eksperimen 8

```
Number: 1Number: 2
Number: 3

Number: 5Number: 4
Number: 6

Number: 9Number: 8
Number: 7

Number: 10
All tasks have been submitted.
```

Eksperimen 9 : Menggunakan Semaphore untuk Membatasi Akses Paralel

```python
import threading
import time

# Membuat semaphore yang hanya mengizinkan 2 thread aktif di waktu yang sama
semaphore = threading.Semaphore(2)

# Fungsi untuk mencetak angka dengan akses terbatas
def print_numbers(n):
    with semaphore:  # Masuk ke dalam semaphore
        print(f"Thread {n} is running")
        time.sleep(2)
        print(f"Thread {n} is done")

# Membuat beberapa thread
threads = []
for i in range(1, 6):
    thread = threading.Thread(target=print_numbers, args=(i,))
    threads.append(thread)

# Memulai semua thread
for thread in threads:
    thread.start()

# Menunggu semua thread selesai
for thread in threads:
    thread.join()

print("All threads have finished execution.")
```

Hasil Running Eksperimen 9

```
Thread 1 is runningThread 2 is running

Thread 1 is doneThread 2 is done
Thread 3 is running

Thread 4 is running
Thread 3 is doneThread 4 is done
Thread 5 is running

Thread 5 is done
All threads have finished execution.
```

Eksperimen 10 : Mengukur Waktu Eksekusi Threading

```python
import threading
import time

def print_numbers():
    for i in range(1, 6):
        time.sleep(1)
        print(f"Number: {i}")

def print_letters():
    for letter in ['A', 'B', 'C', 'D', 'E']:
        time.sleep(1.5)
        print(f"Letter: {letter}")

# Mengukur waktu eksekusi dengan threading
start_time = time.time()

thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)

thread1.start()
thread2.start()

thread1.join()
thread2.join()

end_time = time.time()
print(f"Execution time with threading: {end_time - start_time} seconds")
```

Hasil Running Eksperimen 10

```
Number: 1
Letter: A
Number: 2
Letter: B
Number: 3
Number: 4
Letter: C
Number: 5
Letter: D
Letter: E
Execution time with threading: 7.50463604927063 seconds
```

Kesimpulan

Eksperimen 1: Menambah Thread Tambahan

- Menambahkan lebih banyak thread memungkinkan beberapa tugas dieksekusi secara paralel. Pada eksperimen ini, tiga tugas (angka, huruf, simbol) dijalankan bersamaan, masing-masing dengan delay yang berbeda. Hasilnya menunjukkan bahwa semua output dari ketiga thread tampil secara bergantian sesuai dengan urutan waktu yang berbeda, tanpa saling menunggu.

Eksperimen 2: Mengatur Prioritas dengan Thread.daemon

- hread daemon otomatis berhenti ketika thread utama selesai. Pada eksperimen ini, thread daemon (thread2) mungkin tidak akan menyelesaikan tugasnya karena thread utama (yang menunggu thread1) telah selesai lebih dulu. Ini menunjukkan bahwa daemon thread hanya cocok untuk tugas yang bisa diabaikan jika program utama selesai lebih dulu.

Eksperimen 3: Sinkronisasi dengan Lock

- Dengan menggunakan Lock, kita dapat mengontrol akses ke sumber daya bersama. Pada eksperimen ini, meskipun thread dijalankan secara paralel, output dari setiap thread dikendalikan sehingga tidak ada bentrok atau tumpang tindih dalam mencetak hasil ke terminal. Ini memastikan hanya satu thread yang bisa mencetak di satu waktu.

Eksperimen 4: Menangkap Exception dari Thread

- Mengelola exception dalam thread sangat penting untuk mencegah program crash secara tidak terduga. Pada eksperimen ini, exception ditangkap dengan baik di dalam thread, yang memungkinkan program terus berjalan meskipun terjadi error pada salah satu thread. Ini memastikan bahwa error pada thread individual tidak mempengaruhi keseluruhan aplikasi.

Eksperimen 5: Thread dengan Argumen

- Thread dapat menerima argumen melalui args. Ini memungkinkan fleksibilitas lebih besar dalam mendefinisikan perilaku thread. Dalam eksperimen ini, dua thread menjalankan fungsi yang sama tetapi dengan rentang angka yang berbeda, menunjukkan bahwa kita dapat dengan mudah menjalankan fungsi yang sama secara paralel dengan parameter berbeda.

Eksperimen 6: Menggunakan Queue untuk Komunikasi Antar Thread

- Queue memungkinkan komunikasi yang aman antar-thread. Pada eksperimen ini, thread pertama memasukkan data ke dalam queue, sementara thread kedua mengkonsumsi data tersebut. Hal ini mencegah tumpang tindih akses ke data yang sama dan memastikan komunikasi antar-thread berjalan dengan baik tanpa adanya

-

Eksperimen 7: Menghentikan Thread dengan Event

- Event memungkinkan kita mengontrol kapan sebuah thread harus berhenti. Pada eksperimen ini, thread berhenti secara terkontrol ketika stop_event di-set setelah 5 detik. Ini menunjukkan bagaimana kita bisa dengan aman menginterupsi eksekusi thread yang sedang berjalan.

Eksperimen 8: Menggunakan ThreadPoolExecutor untuk Pooling Thread

- ThreadPoolExecutor mempermudah manajemen thread dengan menjalankan tugas secara paralel menggunakan jumlah thread yang tetap. Pada eksperimen ini, 10 tugas dijalankan menggunakan 3 worker secara paralel, dan manajemen pooling dilakukan secara otomatis oleh executor. Ini sangat berguna dalam kasus di mana banyak tugas kecil perlu dijalankan, menghemat sumber daya daripada membuat terlalu banyak thread.

Eksperimen 9: Menggunakan Semaphore untuk Membatasi Akses Paralel

- Semaphore membatasi jumlah thread yang dapat mengakses bagian kritis kode secara bersamaan. Pada eksperimen ini, hanya 2 thread yang diizinkan untuk berjalan pada saat bersamaan, meskipun ada 5 thread yang dibuat. Ini memastikan bahwa kita bisa mengontrol beban paralel agar tidak terlalu banyak thread yang berjalan sekaligus.

Eksperimen 10: Mengukur Waktu Eksekusi Threading vs Non-threading

- Threading dapat mempercepat eksekusi dalam skenario di mana tugas-tugas dapat berjalan secara paralel tanpa saling menunggu. Pada eksperimen ini, penggunaan threading membuat program selesai lebih cepat dibandingkan non-threading karena kedua fungsi (print angka dan huruf) berjalan bersamaan, dibandingkan dengan non-threading di mana satu fungsi harus menunggu fungsi lainnya selesai.

Untuk Link File Eksperimen :
https://github.com/AgilDeriansyahHasan/Tugas_Pemograman.Paralel.git