Tugas Praktikum

Thread & Task Parallelism

PENDAHULUAN

Modul ini berisi empat tugas praktikum untuk memahami konsep **Thread Parallelism** dan **Task Parallelism** menggunakan Python. Praktikum disusun bertahap mulai dari simulasi sederhana hingga pipeline kompleks, dilengkapi dengan petunjuk, ruang tabel hasil, dan pertanyaan diskusi.

Tugas Praktikum 1 — Thread Parallelism (I/O-Bound Download Simulator)

Tujuan

- Memahami penggunaan thread untuk aplikasi I/O-bound.
- Membandingkan waktu eksekusi serial vs threaded.

Spesifikasi

1. Buat fungsi download_file yang mensimulasikan unduh file dengan sleep.
2. Jalankan 10 file dengan durasi acak (0.5–2 detik).
3. Bandingkan eksekusi **serial** dan **threaded**.
4. Laporkan waktu eksekusi dan speedup.

Starter Code

```
import threading, time, random
def download_file(i, sec):
    time.sleep(sec)
def run_serial(jobs):
    for i, sec in jobs: download_file(i, sec)
def run_threads(jobs):
    threads=[]
    for i, sec in jobs:
        t=threading.Thread(target=download_file,args=(i,sec))
        threads.append(t); t.start()
    for t in threads: t.join()
```

Tabel Hasil

Mode	Jumlah File	Waktu (s)	Speedup

Pertanyaan Diskusi

- Mengapa threads lebih efektif untuk aplikasi I/O dibanding serial?
- Apa potensi masalah jika terlalu banyak threads dibuat?

Tugas Praktikum 2 — Task Parallelism (CPU-Bound Heavy Computation)

Tujuan

- Memahami penggunaan **ProcessPoolExecutor** untuk CPU-bound.
- Menghitung speedup dan efisiensi.

Spesifikasi

1. Definisikan fungsi heavy(n) dengan loop 10**6.
2. Jalankan **serial** dan **parallel** (2,4,8 proses).
3. Hitung speedup dan efisiensi.

Starter Code

```
from concurrent.futures import ProcessPoolExecutor
def heavy(n, iters=10**6):
    s=0
    for i in range(iters): s+=(i*n)%7
    return s
with ProcessPoolExecutor(max_workers=4) as ex:
    ex.map(heavy, range(1,9))
```

Tabel Hasil

Proses	Waktu (s)	Speedup	Efisiensi

Pertanyaan Diskusi

- Mengapa task parallelism lebih efektif untuk CPU-bound dibanding thread?
- Apa peran **GIL (Global Interpreter Lock)** dalam Python?

Tugas Praktikum 3 — Perbandingan Threads vs Tasks

Tujuan

- Membandingkan performa threads dan processes untuk I/O-bound & CPU-bound.

Spesifikasi

1. Gunakan fungsi dari **Tugas 1** (I/O) dan **Tugas 2** (CPU).
2. Jalankan **serial**, **threads**, dan **processes**.
3. Buat tabel hasil perbandingan.
4. Analisis mengapa threads unggul di I/O, processes unggul di CPU.

Starter Code

```
# Jalankan serial, threads, dan processes
# Catat waktu dan buat tabel perbandingan
```

Tabel Hasil

Jenis Aplikasi	Serial (s)	Threads (s)	Processes (s)	Speedup Threads	Speedup Processes

Pertanyaan Diskusi

- Jelaskan perbedaan hasil antara aplikasi I/O-bound dan CPU-bound.
- Apa implikasi pemilihan threads vs processes pada desain sistem nyata?

Tugas Praktikum 4 — Hybrid Pipeline (Threads + Processes)

Tujuan

- Mendesain **pipeline multi-tahap** dengan threads (I/O) dan processes (CPU).
- Mengukur throughput dan latensi.

Spesifikasi

1. Siapkan dataset (teks/gambar).
2. **Stage-1**: loader threads membaca path file ke queue.
3. **Stage-2**: process pool mengolah isi file.
4. **Stage-3**: agregasi hasil.
5. Ukur waktu total, throughput, dan latensi.
6. Bandingkan dengan baseline single-process.

Starter Code

```
import threading, queue
from concurrent.futures import ProcessPoolExecutor
# Stage-1: loader thread -> queue
# Stage-2: process pool -> compute
# Stage-3: aggregator -> merge results
```

Tabel Hasil

Threads Loader	Workers CPU	Jumlah File	Waktu (s)	Throughput (file/s)	Avg Latency (s)

Pertanyaan Diskusi

- Dimana bottleneck pipeline terjadi?
- Bagaimana peran **backpressure (queue maxsize)** dalam pipeline ini?
- Bagaimana menyesuaikan jumlah thread vs proses untuk optimasi?

PENUTUP

Dengan menyelesaikan keempat praktikum ini, mahasiswa akan:

1. Memahami perbedaan **Thread Parallelism** dan **Task Parallelism**.
2. Mengetahui konteks aplikasi I/O-bound vs CPU-bound.
3. Dapat merancang pipeline hibrid untuk kasus nyata.