MODUL PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

Minggu 7

KONKURENSI

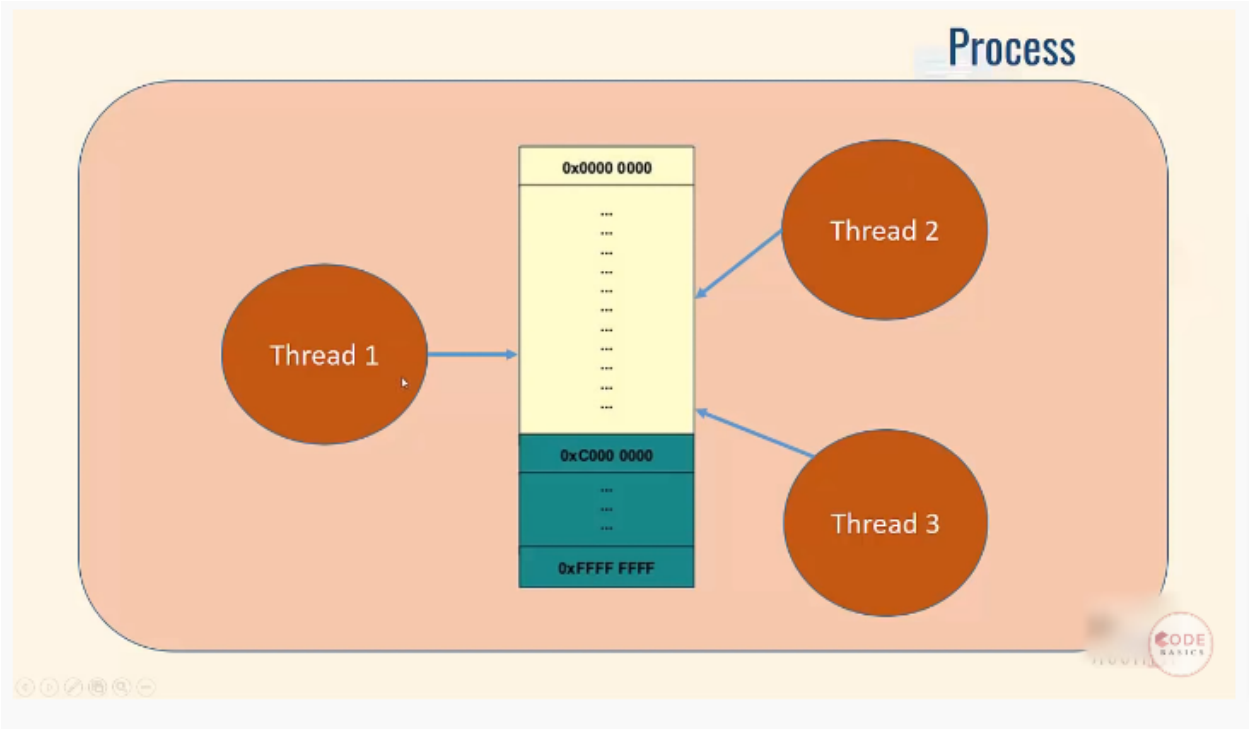
Tim Pengajar:
Ahmad Luky Ramdani S.Komp., M.Kom
Arief Ichwani S.Kom., M.Cs.
Eko Dwi Nugroho, S.Kom., M.Cs.
Muhammad Habib Algifari, S.Kom. M.T.I.
Lukas Sandy
Aminudin Fadila
Kharisma Anjina
Vina Alvionita

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK ELEKTRO, INFORMATIKA DAN SISTEM FISIS
INSTITUT TEKNOLOGI SUMATERA
2021

DAFTAR ISI

1. Threading

Threading adalah salah satu cara bagaimana kita dapat melakukan konkurensi dalam mengeksekusi sebuah operasi. Tidak seperti kode tanpa *threading* yang harus menunggu proses eksekusi kode sebelumnya beres, *threading* memisahkan sebagian kode dan mengeksekusinya di proses yang dia ciptakan sendiri.



Dalam python, untuk menjalankan konsep multithreading, digunakan module threading. Dan dalam pengimplementasiannya, dapat dilakukan dengan 2 cara, yaitu dengan instansiasi class Thread, dan inheritance class Thread.

```python
import time
#tanpa konkurensi
class olah_data:
  def __init__(self, rentang):
    self.rentang = rentang

  def ambil(self):
    print(f'[1] ambil dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[1] ambil dari data ke : {self.rentang}  selesai')
  def sortir(self):
    print(f'[2] sortir dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[2] sortir dari data ke : {self.rentang}  selesai')
  def export(self):
    print(f'[3] export dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[3] export dari data ke : {self.rentang}  selesai ')
  def run(self):
    self.ambil()
    self.sortir()
    self.export()

rents = [
        '1 - 10000', '10001 - 20000', '20001 - 30000'
]


start = time.perf_counter()
```

```python
for rentang in rents:
  olah_data(rentang).run()

finish = time.perf_counter()
print(f'selesai dalam {finish - start} detik')
```

```
[1] ambil dari data ke : 1 - 10000
[1] ambil dari data ke : 1 - 10000  selesai
[2] sortir dari data ke : 1 - 10000
[2] sortir dari data ke : 1 - 10000  selesai
[3] export dari data ke : 1 - 10000
[3] export dari data ke : 1 - 10000  selesai
[1] ambil dari data ke : 10001 - 20000
[1] ambil dari data ke : 10001 - 20000  selesai
[2] sortir dari data ke : 10001 - 20000
[2] sortir dari data ke : 10001 - 20000  selesai
[3] export dari data ke : 10001 - 20000
[3] export dari data ke : 10001 - 20000  selesai
[1] ambil dari data ke : 20001 - 30000
[1] ambil dari data ke : 20001 - 30000  selesai
[2] sortir dari data ke : 20001 - 30000
[2] sortir dari data ke : 20001 - 30000  selesai
[3] export dari data ke : 20001 - 30000
[3] export dari data ke : 20001 - 30000  selesai
selesai dalam 18.02296076300013 detik
```

Jika tidak menggunakan Threading, diperlukan 18 detik untuk melakukan operasi tersebut.

```python
#dengan multithreading menggunakan cara instansiasi

from threading import Thread
import time

class olah_data:
  def __init__(self, rentang):
    self.rentang = rentang

  def ambil(self):
    print(f'[1] ambil dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[1] ambil dari data ke : {self.rentang}  selesai')
  def sortir(self):
    print(f'[2] sortir dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[2] sortir dari data ke : {self.rentang}  selesai')
  def export(self):
    print(f'[3] export dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[3] export dari data ke : {self.rentang}  selesai ')
  def run(self):
    self.ambil()
    self.sortir()
    self.export()

rents = [
        '1 - 10000',
         '10001 - 20000',
         '20001 - 30000'
]
```

```python
start = time.perf_counter()
for rentang in rents:
  # olah_data(rentang).run()
  t = Thread(target=olah_data(rentang).run)
  t.start()
t.join()
finish = time.perf_counter()
print(f'selesai dalam {finish - start} detik')
```

```
[1] ambil dari data ke : 1 - 10000
[1] ambil dari data ke : 10001 - 20000
[1] ambil dari data ke : 20001 - 30000
[1] ambil dari data ke : 1 - 10000  selesai
[2] sortir dari data ke : 1 - 10000
[1] ambil dari data ke : 10001 - 20000  selesai
[2] sortir dari data ke : 10001 - 20000
[1] ambil dari data ke : 20001 - 30000  selesai
[2] sortir dari data ke : 20001 - 30000
[2] sortir dari data ke : 1 - 10000  selesai
[3] export dari data ke : 1 - 10000
[2] sortir dari data ke : 10001 - 20000  selesai
[3] export dari data ke : 10001 - 20000
[2] sortir dari data ke : 20001 - 30000  selesai
[3] export dari data ke : 20001 - 30000
[3] export dari data ke : 1 - 10000  selesai
[3] export dari data ke : 10001 - 20000  selesai
[3] export dari data ke : 20001 - 30000  selesai
selesai dalam 6.012457044999792 detik
```

```python
#dengan multithreading menggunakan cara inheritence
from threading import Thread
import time

class olah_data(Thread):
  def __init__(self, rentang):
    self.rentang = rentang
    super().__init__()
  def ambil(self):
    print(f'[1] ambil dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[1] ambil dari data ke : {self.rentang}  selesai')
  def sortir(self):
    print(f'[2] sortir dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[2] sortir dari data ke : {self.rentang}  selesai')
  def export(self):
    print(f'[3] export dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[3] export dari data ke : {self.rentang}  selesai ')
  def run(self):
    self.ambil()
    self.sortir()
    self.export()

rents = [
        '1 - 10000', '10001 - 20000', '20001 - 30000'
]
```

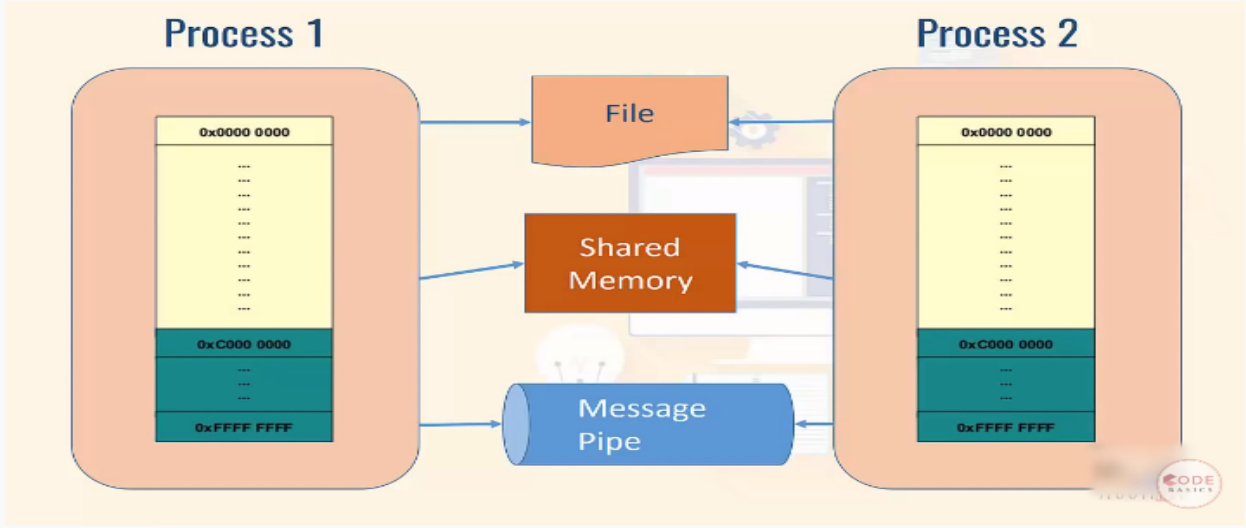
```python
start = time.perf_counter()
for rentang in rents:
  # olah_data(rentang).run()
  t = olah_data(rentang)
  t.start()
t.join()
finish = time.perf_counter()
print(f'selesai dalam {finish - start} detik')
```

```
[1] ambil dari data ke : 1 - 10000[1] ambil dari data ke : 10001 - 20000

[1] ambil dari data ke : 20001 - 30000
[1] ambil dari data ke : 10001 - 20000  selesai
[2] sortir dari data ke : 10001 - 20000
[1] ambil dari data ke : 20001 - 30000  selesai
[2] sortir dari data ke : 20001 - 30000
[1] ambil dari data ke : 1 - 10000  selesai
[2] sortir dari data ke : 1 - 10000
[2] sortir dari data ke : 10001 - 20000  selesai[2] sortir dari data ke : 1 - 10000  selesai
[3] export dari data ke : 1 - 10000
[2] sortir dari data ke : 20001 - 30000  selesai
[3] export dari data ke : 10001 - 20000

[3] export dari data ke : 20001 - 30000
[3] export dari data ke : 1 - 10000  selesai
[3] export dari data ke : 10001 - 20000  selesai
[3] export dari data ke : 20001 - 30000  selesai
selesai dalam 6.0099235819998285 detik
```

Dengan menggunakan Threading waktu yg diperlukan menjadi 6 detik.

2. Multiprocessing

Sama seperti multithreading, multiprocess juga dibuat untuk mengimplementasikan konsep konkurensi pada bahasa pemrograman python, perbedaannya adalah thread berjalan di satu core/cpu yang sama, sedangkan process dilakukan di core/cpu yg berbeda. Sehingga berbagi data pada multiprocessing adalah sesuatu yang tricky



```python
#dengan multiprocessing
from multiprocessing import Process
import time
class olah_data:
  def __init__(self, rentang):
    self.rentang = rentang
  def ambil(self):
    print(f'[1] ambil dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[1] ambil dari data ke : {self.rentang}  selesai')
  def sortir(self):
    print(f'[2] sortir dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[2] sortir dari data ke : {self.rentang}  selesai')
  def export(self):
    print(f'[3] export dari data ke : {self.rentang}')
    time.sleep(2)
    print(f'[3] export dari data ke : {self.rentang}  selesai ')
  def run(self):
    self.ambil()
    self.sortir()
    self.export()
rents = [
        '1 - 10000', '10001 - 20000', '20001 - 30000'
]
```

```python
start = time.perf_counter()
for rentang in rents:
  # olah_data(rentang).run()
  t = Process(target=olah_data(rentang).run)
  t.start()
  time.sleep(1)
t.join()
finish = time.perf_counter()
print(f'selesai dalam {finish - start} detik')
```

```
[1] ambil dari data ke : 1 - 10000
[1] ambil dari data ke : 10001 - 20000
[1] ambil dari data ke : 1 - 10000  selesai
[2] sortir dari data ke : 1 - 10000
[1] ambil dari data ke : 20001 - 30000
[1] ambil dari data ke : 10001 - 20000  selesai
[2] sortir dari data ke : 10001 - 20000
[2] sortir dari data ke : 1 - 10000  selesai
[3] export dari data ke : 1 - 10000
[1] ambil dari data ke : 20001 - 30000  selesai
[2] sortir dari data ke : 20001 - 30000
[2] sortir dari data ke : 10001 - 20000  selesai
[3] export dari data ke : 10001 - 20000
[3] export dari data ke : 1 - 10000  selesai
[2] sortir dari data ke : 20001 - 30000  selesai
[3] export dari data ke : 20001 - 30000
[3] export dari data ke : 10001 - 20000  selesai
[3] export dari data ke : 20001 - 30000  selesai
selesai dalam 8.057074753000052 detik
```

Menggunakan multithreading atau multiprocessing?
Program GUI menggunakan threading setiap saat untuk membuat aplikasi menjadi responsif. Misalnya, dalam program teks editor, satu thread dapat merekam input pengguna, thread lainnya bertanggung jawab untuk menampilkan teks, thread ketiga dapat melakukan pemeriksaan ejaan, dan seterusnya. Di sini, program harus menunggu interaksi pengguna, yang merupakan hambatan terbesar. Menggunakan multiprosesing tidak akan membuat program menjadi lebih cepat.
Kasus penggunaan lain untuk threading adalah program yang dipengaruhi IO atau terikat jaringan, seperti web scraper. Dalam kasus ini, beberapa thread dapat scrape beberapa page web secara paralel. Thread harus mengunduh halaman web dari Internet, dan itu akan menjadi hambatan terbesar, jadi threading adalah solusi sempurna di sini. Server web, karena terikat jaringan, bekerja dengan cara yang sama; dengan mereka, multiprocess tidak memiliki keunggulan apa pun dibandingkan threading. Contoh lain yang relevan adalah Tensorflow, yang menggunakan kumpulan thread untuk mengubah data secara paralel.

Multiprocessing mengalahkan threading jika program membutuhkan banyak CPU dan tidak perlu melakukan IO atau interaksi pengguna. Misalnya, program apa pun yang hanya mengolah angka akan melihat percepatan besar-besaran dari multiprocessing; faktanya, threading mungkin akan memperlambatnya. Contoh dunia nyata yang menarik adalah Pytorch Dataloader, yang menggunakan banyak subproses untuk memuat data ke GPU.

3. AsyncIO

AsyncIO diperkenalkan dalam python mulai versi 3.4, dan terus berkembang, pada modul ini digunakan python versi 3.7. Jika python praktikan dibawah 3.7 silahkan untuk menginstal versi ketujuh atau atasnya.

asyncIO dijalankan dengan single-threaded dan single-process, asyncIO bukan bagian multithreading maupun multiprocess, bahkan asyncIO juga bukan paralel. asyncIO menggunakan konsep cooperative multitasking. Dengan kata lain, asyncIO memberikan rasa konkurensi walaupun hanya dengan 1 thread di 1 core CPU.

Async IO mungkin pada awalnya tampak berlawanan dengan intuisi dan paradoks. Bagaimana sesuatu yang memfasilitasi kode bersamaan menggunakan satu utas dan satu inti CPU?
Penjelasan dengan contoh di real-world:

> Master catur Judit Polgár menyelenggarakan pameran catur di mana dia memainkan banyak pemain amatir. Dia memiliki dua cara untuk menyelenggarakan pameran: sinkron dan asinkron.
>
> Asumsi:
> - 24 lawan
> - Judit membuat setiap gerakan catur dalam 5 detik
> - Lawan masing-masing membutuhkan waktu 55 detik untuk bergerak
> - Rata-rata permainan 30 pasangan gerakan (total 60 gerakan)
>
> Versi sinkron : Judit memainkan satu game pada satu waktu, tidak pernah dua game pada saat yang sama, hingga game tersebut selesai. Setiap permainan membutuhkan waktu (55 + 5) * 30 == 1800 detik, atau 30 menit. Seluruh pameran membutuhkan waktu 24 * 30 == 720 menit, atau 12 jam .
>
> Versi asinkron : Judit berpindah dari meja ke meja, membuat satu gerakan di setiap meja. Dia meninggalkan meja dan membiarkan lawan melakukan langkah selanjutnya. Satu gerakan di semua 24 game Judit membutuhkan 24 * 5 == 120 detik, atau 2 menit. Seluruh pameran sekarang dipotong menjadi 120 * 30 == 3600 detik, atau hanya 1 jam .

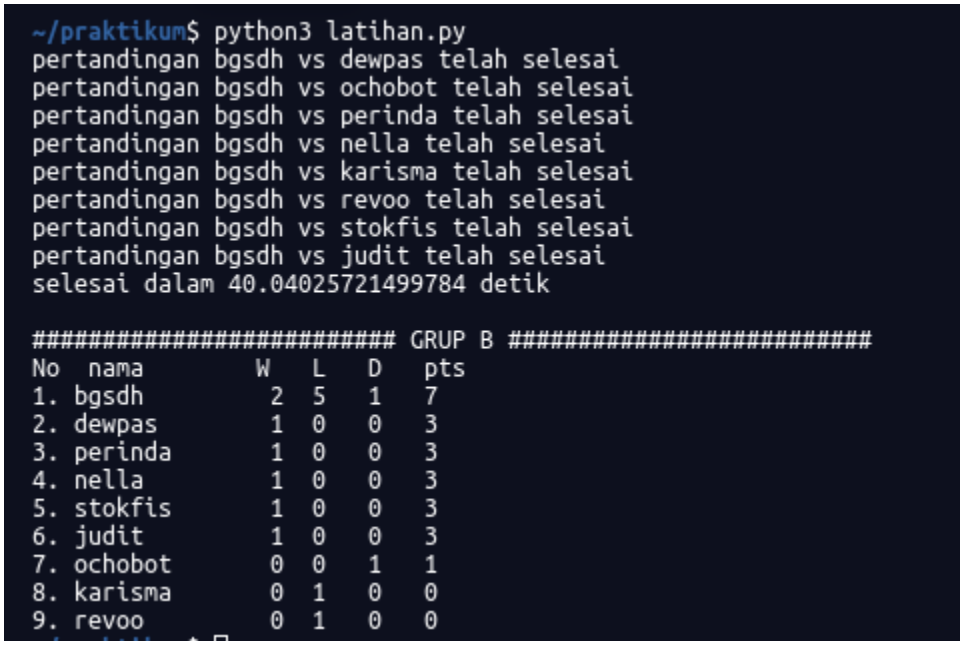
Untuk mengimplementasikan asyncIO, cukup berbeda dengan thread dan process. Berikut adalah program dari kode-kode yg ada pada subbab multithreading dan multiprocessing, diimplementasikan dengan asyncIO

```python
import asyncio
import time
class Dapur:
  async def masak(self,masakan,waktu):
    print(f'pesanan {masakan} diterima')
    await asyncio.sleep(waktu)
    print(f'{masakan} selesai dimasak')
class Pramusaji:
  def __init__(self):
    self.dapur = Dapur()
  async def pesan(self):
    t1 = asyncio.create_task(
      self.dapur.masak('nasi goreng sipud',5))
    t2 = asyncio.create_task(
      self.dapur.masak('es kelapa abg',2))
    t3 = asyncio.create_task(
      self.dapur.masak('paket mahasiswa',3))
    t4 = asyncio.create_task(
      self.dapur.masak('teh es',1))
    await t1
    await t2
    await t3
    await t4
p = Pramusaji()
asyncio.run(p.pesan())
```

```
~/praktikum$ python3 asyncrev1.py
pesanan nasi goreng sipud diterima
pesanan es kelapa abg diterima
pesanan paket mahasiswa diterima
pesanan teh es diterima
teh es selesai dimasak
es kelapa abg selesai dimasak
paket mahasiswa selesai dimasak
nasi goreng sipud selesai dimasak
~/praktikum$
```

Latihan

Lihat kode pada link berikut https://pastebin.com/79ixZF2A, program tersebut menjalankan pertandingan catur secara sekuensial/secara berurutan. Berikut hasil outputnya:



```
~/praktikum$ python3 latihan.py
pertandingan bgsdh vs dewpas telah selesai
pertandingan bgsdh vs ochobot telah selesai
pertandingan bgsdh vs perinda telah selesai
pertandingan bgsdh vs nella telah selesai
pertandingan bgsdh vs karisma telah selesai
pertandingan bgsdh vs revoo telah selesai
pertandingan bgsdh vs stokfis telah selesai
pertandingan bgsdh vs judit telah selesai
selesai dalam 40.04025721499784 detik

########################## GRUP B ##########################
No  nama        W   L   D   pts
1. bgsdh         2  5   1   7
2. dewpas        1  0   0   3
3. perinda       1  0   0   3
4. nella         1  0   0   3
5. stokfis       1  0   0   3
6. judit         1  0   0   3
7. ochobot       0  0   1   1
8. karisma       0  1   0   0
9. revoo         0  1   0   0
```

Diperlukan waktu 40 detik untuk menyelesaikan program tersebut.

Edit lah kode tersebut, gunakan modul multithreading,multiprocessing dan asyncIO sehingga membuat waktu yg dibutuhkan menjadi lebih singkat.

(untuk mempermudah praktikan,

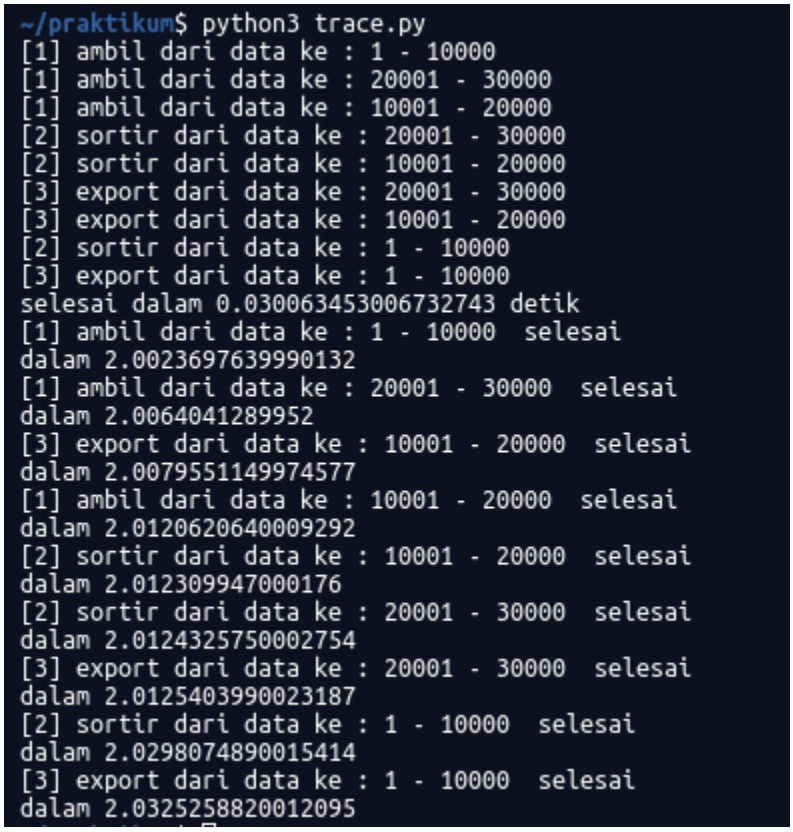
- Nim mod 3 = 0 mengerjakan menggunakan threading
- Nim mod 3 = 1 mengerjakan menggunakan multiprocessing
- Nim mod 3 = 2 mengerjakan menggunakan asyncIO

)

Tugas

1. Modifikasi kode pada subbab Threading yang tanpa konkurensi, gunakan modul asyncIO untuk membuatnya lebih cepat.

2. Kode pada bab Threading dan Multiprocessing masih bisa ditingkatkan efisiensinya, dengan cara menggunakan threading/processing juga didalam method run nya
   Contoh output Threading



Contoh output multiprocessing

```
~/praktikum$ python3 trace.py
[1] ambil dari data ke : 1 - 10000
[1] ambil dari data ke : 10001 - 20000
[2] sortir dari data ke : 10001 - 20000
[2] sortir dari data ke : 1 - 10000
[3] export dari data ke : 1 - 10000
[3] export dari data ke : 20001 - 30000
[3] export dari data ke : 10001 - 20000
[2] sortir dari data ke : 20001 - 30000
[1] ambil dari data ke : 20001 - 30000
[1] ambil dari data ke : 1 - 10000  selesai
[1] ambil dari data ke : 10001 - 20000  selesai
[2] sortir dari data ke : 10001 - 20000  selesai
[2] sortir dari data ke : 1 - 10000  selesai
[3] export dari data ke : 1 - 10000  selesai
[3] export dari data ke : 20001 - 30000  selesai
[3] export dari data ke : 10001 - 20000  selesai
[2] sortir dari data ke : 20001 - 30000  selesai
[1] ambil dari data ke : 20001 - 30000  selesai
selesai dalam 2.0278785190021154 detik
```

3. Terdapat kesalahan (secara konsep, bukan secara code) pada kode jawaban dari soal tugas nomor 2, jelaskan kesalahan yang terjadi !

Referensi

- Slide perkuliahan pemrograman berbasis objek minggu 12
- realpython.com/async-io-python/
- Python.readthedocs.io
- Slamet, Rifai. "Learn Python Threading to run multiple programs at once || Python Threading Tutorial" YouTube, diunggah oleh Rifai Slamet,