Facultatea Calculatoare, Informatica si Microelectronica

Universitatea Tehnica a Moldovei

Programarea in retea

Lucrarea de laborator#2

# HTTP Client with concurrency superpowers

*Autor:*
Alexandru Stamatin

*lector asistent:*
Alexandru Gavrisco

# Lucrarea de laborator #2

## 1 Scopul lucrarii de laborator

Realizarea unui client HTTP utilizand tehnicile programarii concurente

## 2 Obiective

- Studierea modelului OSI
- Studierea serializarii/deserializarii
- Studierea concurentei

# 3 Efectuarea lucrarii de laborator

## 3.1 Sarcinile propuse

- Obtinerea datelor despre categorii si produse in mod concurent
- Parsarea si validarea datelor receptionate
- Agregarea concurenta a datelor
- Afisarea si stocarea locala a datelor in mod concurent

## 3.2 Analiza lucrarii de laborator

In cadrul acestei lucrari de laborator am dezvoltat o aplicatie care primeste date cu privire la comenzi si categoriile de produse de la un API, realizand apoi agregarea si afisarea datelor intr-un format comod pentru utilizator. Primirea datelor, agregarea, afisarea si salvarea locala sunt realizat in mod concurent.

- Primirea datelor

  Deoarece apelarea metodei requestes.get() blocheaza firul de executie, executia acesteia este delegata Thread pool-ului implicit utilizand loop.run_in_executor cu primul argument None.

```
async def getData(loop):
    def categ():
        try:
            return requests.get('https://evil-legacy-service.herok
        except:
            print("Cannot connect to server. Showing cached data")
            print(pd.read_pickle('localcache'))
            sys.exit(1)
    def ord():
        try:
            return requests.get(order,headers=headers, params=data
        except:
            pass
    future1 = loop.run_in_executor(None, categ)
    future2 = loop.run_in_executor(None, ord)
    response1 = await future1
    response2 = await future2
    return response1, response2
```

- Agregarea

  Pentru agregarea datelor a fost utilizat pachetul pandas. Datele primite in format csv au fost transformate in liste de liste. In continuare au fost create DataFrame-uri pentru categorii si liste, asupra carora a fost aplicata operatiunea merge. Asupra DataFrame-ului obtinut a fost aplciata metoda groupby si sum pentru a calcula numarul de comenzi pentru fiecare categorie.

```
        Orders = pd.DataFrame(ListOrd, columns=['id', 'total', 'id
        Categs = pd.DataFrame(ListCateg, columns=['id_cat', 'Categ

#       Set type of id_cat column to integer

        Categs['id_cat'] = Categs['id_cat'].astype(int)
        Orders['id_cat'] = Orders['id_cat'].astype(int)

#       Merge Orders and Categs tables. Orders grouped by Category

        merged = Categs.merge(Orders,how='left')
        merged = pd.to_numeric(merged.total).groupby([merged.id_ca
        merged = merged.to_frame()
```

  In continuare la categoriile parinte au fost adaugate comenzile de la categoriile copil. Acest lucru a fost realizat in 2 etape: la inceput au fost adaugata comenzile de la categoriile fara copii la cele cu un copil, apoi de la cele cu un copil la cele cu 2 copii.

```
#       Calculate the number of parents for each Category

        for cat in cats:
            val = cats[cat]
            while val != '':
                parent[int(cat)] += 1
                val = cats[val]

#       Determine maximum number of parents

        maxparent = max(parent,key=parent.get)
        maxparents = parent[maxparent]


#       Aggregate data. Orders from child categories are added to
```

```
        for i in range(maxparents,0,-1):
            for tup in merged.itertuples():
                toadd[int(tup[0][0])] += float(tup[1])
                if (tup[0][2] != '') and (parent[tup[0][0]] == i):
                    toadd[int(tup[0][2])] += float(tup[1])

            for tup in merged.itertuples():
                merged.loc[int(tup[0][0]), 'total'] = toadd[int(tu
            toadd.update((k, 0) for k in toadd)
```

- Afisarea/Salvarea

  Datele primite sunt salvate local si afisate in mod concurent

```
async def CacheDisplay(loop):
    def Savefile():
        merged.to_pickle('localcache')
    def File():
        print(merged)
    future = loop.run_in_executor(None, File)
    future1 = loop.run_in_executor(None, Savefile)
    response1 = await future1
    response = await future
    return response, response1
```

| id_cat | Category | Parent_Category | |
|---|---|---|---|
| 1 | Computers | | 165.62 |
| 2 | Laptops | 1 | 8.11 |
| 3 | Network Accessories | 1 | 5.25 |
| 4 | Tablets | 1 | 0.00 |
| 5 | PC Components | 1 | 152.26 |
| 6 | CPU | 5 | 0.00 |
| 7 | GPU | 5 | 5.50 |
| 8 | RAM | 5 | 37.32 |
| 9 | SSD/HDD | 5 | 93.76 |
| 10 | Motherboard | 5 | 0.00 |
| 11 | Electronics | | 1082.12 |
| 12 | TV | 11 | 1.89 |
| 13 | Headphones | 11 | 0.00 |
| 14 | Wearables | 11 | 914.47 |
| 15 | Smartwatches | 14 | 2.42 |
| 16 | VR/AR | 14 | 8.00 |
| 17 | Action Cameras | 14 | 0.00 |
| 18 | Activity Trackers | 14 | 0.00 |
| 19 | Photo & Video | 11 | 0.00 |
| 20 | Automotive | | 1155.40 |
| 21 | Tires | 20 | 0.00 |
| 22 | Wheels | 20 | 55.94 |
| 23 | GPS & Cameras | 20 | 882.00 |
| 24 | Food & Grocery | | 0.00 |

Figure 1: Rezultatul rularii aplicatiei

# Concluzie

Pentru realizarea sarcinilor propuse in cadrul lucrarii a fost studiat procesul de implementare a tehnicilor programarii concurente in limbajul python utilizand pachetul asyncio. Metodele ce blocheaza firul de executie au fost rulate intr-un thread pool separat utilizand metoda run_in_executor iar corutinele au fost executate prin metoda run_until_complete. Pentru agregarea datelor am studiat pachetul pandas ce permite realizarea diferitor operatiuni cu tabele. Metodele sum si groupby au fost utilizate pentru afisarea datelor intr-un format comod pentru utilizator.

# References

[1] Luciano Ramalho. *Fluent Python*.

[2] Andrew S. Tanenbaum. *Computer Networks*