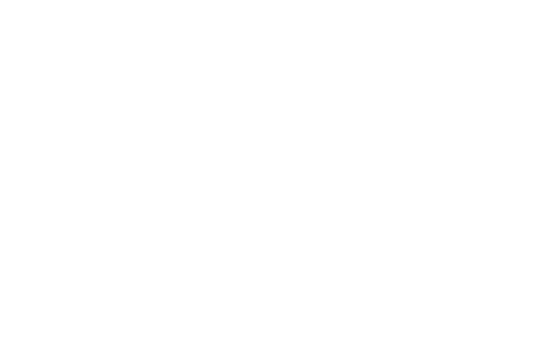
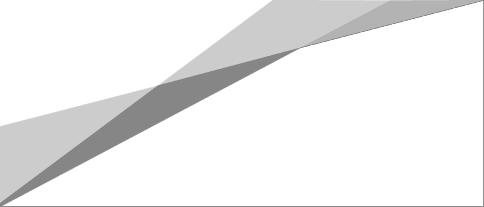
KOLEGIJ

***Mrežno programiranje***

Laboratorijska vježba br.6

***Teme:***

• Multhreading



U ovom laboratorijskoj vježbi ćemo proći implementaciju multithreading principa u Pythonu.

**ŠTO JE VIŠENITNOST**

**Multithreading ili višenitnost** je sposobnost centralne procesorske jedinice, tj. CPU-a ili pak jedne jezgre u višejezgrenim procesorima da izvrše višestruke procese ili threadove, tj. niti istovremeno. Za razliku od višeprocesnih aplikacija, u višenitnim aplikacijama niti dijele resurse jedne jezgre ili CPU-a.

Pokretanje nekoliko niti je slično pokretanju nekoliko različitih programa istovremeno, ali sa sljedećim prednostima:

• Višestruke niti unutar procesa dijeli isti podatkovni prostor s glavnom niti, te stoga mogu dijeliti informacije ili komunicirati jedna s drugom na jednostavniji način nego što bi to mogli da su odvojeni procesi

• Niti ili threadovi se ponekad nazivaju laganim, tj. *lightweight* procesima i ne zahtijevaju previše memorije što znači da su u resursnom pogledu jeftiniji od procesa.

Nit ima početak, egzekucijsku sekvencu i završetak.

**RAD S NITIMA:**

***THREAD* MODUL**

Da bi radili s nitima prvo trebamo uvesti **thread** modul:

**import** thread

Za pokretanje nove niti pozivamo sljedeću metodu iz thread modula:

thread**.**start\_new\_thread **(** function**,** args**[,** kwargs**] )**

Poziv ove metode omogućuje kreiranje nove niti na brz i efikasan način i u Linux i u Windows okruženju.

Poziv metode se izvršava odmah, nova *child* nit se kreira i poziva funkciju s proslijeđenom listom argumenata. Kada se funkcija izvrši nit se ukida.

U ovoj metodi *agrs* je tzv. tuple (sekvenca nepromjenjivih Python objekata) argumenata. Prazni tuple se koristi za poziv funkcije koja ne prima argumente.

**PRIMJER**

Pokrenite i vidite rezultat.

#threading\_primjer\_.py

#Mrežno programiranje LABno6 2019

**import** thread

**import** time

# Definicija funkcije za nit

**def** print\_time**(** threadName**,** delay**):**

count **=** 0

**while** count **<** 5**:**

time**.**sleep**(**delay**)**

count **+=** 1

**print** "%s: %s" **% (** threadName**,** time**.**ctime**(**time**.**time**()) )**

# Kreiramo dvije niti

**try:**

thread**.**start\_new\_thread**(** print\_time**, (**"Thread-1"**,** 2**, ) )**

thread**.**start\_new\_thread**(** print\_time**, (**"Thread-2"**,** 4**, ) )**

**except:**

**print** "Greška: ne mogu pokrenuti nit!!"

# Čekaj dok se sve niti ne izvrše

**while** 1**:**

**pass**

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

Iako je ovaj gore modul vrlo efikasan za jednostavno upravljanje nitima postoji i noviji modul u Pythonu imena *threading* u odnosu na kojeg je *thread* modul jako ograničen.

***THREADING* MODUL**

Ovaj modul omogućava mnogo moćnije i efikasnije upravljanje nitima. Ovaj modul pruža sve metode kao i *thread* modul, te pruže neke dodatne metode:

• **threading.activeCount()** – vraća broj thread objekata koji su

aktivni.

• **threading.currentThread()** – vraća trenutni thread objekt.

• **threading.enumerate()** – vraća listu svih thread objekata koji su

trenutno aktivni.

Dodatno na ove metode, *threading* modul ima *Thread* klasu koja implementira niti. Metode koje pruža ta klasa su:

• **run()** – ulazna točka za nit.

• **start()** – pokreće nit pozivajući run() metodu

• **join([time])** – čeka nit da završi

• **isAlive()** – provjerava da li se nit još izvršava

• **getName()** – vraća ime niti

• **setName()** – postavlja ime niti

Da bi implementirali novu nit koristeći threading modul moramo napraviti sljedeće:

o Definirati novu podklasu klase *Thread*

o Override-ati *\_init\_(self [,args])* metodu da se može dodati još

argumenata

o Override-ati *run(self [,args])* metodu da bi imeplemetirali što

nit mora napraviti kada se pokrene.

Kad kreirate novu *Thread* podklasu, možete kreirati njenu instancu, te onda pokrenuti novi thread pokrećući *start()* metodu, koja onda poziva *run()* metodu.

**PRIMJER**

#threading\_primjer\_threading\_modul.py

#Mrezno programiranje LABno6 2019

**import** threading

**import** time exitFlag **=** 0

**class myThread (**threading**.**Thread**):**

**def** init **(**self**,** threadID**,** name**,** counter**):** threading**.**Thread**.** init **(**self**)** self**.**threadID **=** threadID

self**.**name **=** name self**.**counter **=** counter

**def** run**(**self**):**

**print** "Pokrecem nit " **+** self**.**name

print\_time**(**self**.**name**,** 5**,** self**.**counter**)**

**print** "Izlazim iz niti " **+** self**.**name

**def** print\_time**(**threadName**,** counter**,** delay**):**

**while** counter**:**

**if** exitFlag**:**

threadName**.**exit**()**

time**.**sleep**(**delay**)**

**print** "%s: %s" **% (**threadName**,** time**.**ctime**(**time**.**time**()))**

counter **-=** 1

# Kreiraj nove niti

thread1 **=** myThread**(**1**,** "Thread-1"**,** 1**)**

thread2 **=** myThread**(**2**,** "Thread-2"**,** 2**)**

# Pokreni nove niti thread1**.**start**()** thread2**.**start**()**

**print** "\nIzlazim iz glavne niti\n

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

**SINKRONIZACIJA NITI**

Threading modul uključuje jednostavan mehanizam za zaključavanje niti koji omogućuje sinkronizaciju niti. Novi lock se kreira pozivanjem **Lock()** metode koja vraća novi lock.

**acquire(blocking)** metoda novog lock objekta se koristi u svrhu prisiljavanja niti da se izvršavaju sinkrono. Opcinalan *blocking* parametar omogućuje kontrolu da li nit čeka da bi ostvarila lock.

Ako je *blocking* postavljen na 0 nit se smjesta vraća s vrijednošću 0 ako lock ne može biti ostvaren, ili vraća 1 ako je lock ostvaren. Ako je blocking postavljen na 1, nit blokira i čeka da se lock oslobodi.

**release()** metoda novog lock objekta se koristi za otpuštanje locka kada više nije potreban.

**PRIMJER**

#threading\_primjer\_sinkronizacija.py

#Mrezno programiranje LABno6 2019

**import** threading

**import** time

**class myThread (**threading**.**Thread**):**

**def** init **(**self**,** threadID**,** name**,** counter**):**

threading**.**Thread**.** init **(**self**)** self**.**threadID **=** threadID self**.**name **=** name

self**.**counter **=** counter

**def** run**(**self**):**

**print** "Pokrecem nit " **+** self**.**name

# Ostvari lock zbog sinkronizacije niti

threadLock**.**acquire**()**

print\_time**(**self**.**name**,** self**.**counter**,** 3**)**

# Oslobodi lock da bi se izvrsila sljedeca nit

threadLock**.**release**()**

**def** print\_time**(**threadName**,** delay**,** counter**):**

**while** counter**:**

time**.**sleep**(**delay**)**

**print** "%s: %s" **% (**threadName**,** time**.**ctime**(**time**.**time**()))**

counter **-=** 1

threadLock **=** threading**.**Lock**()**

threads **= []**

# Kreiraj nove niti

thread1 **=** myThread**(**1**,** "Thread-1"**,** 1**)**

thread2 **=** myThread**(**2**,** "Thread-2"**,** 2**)**

# Pokreni nove niti thread1**.**start**()** thread2**.**start**()**

# Dodaj niti u thread listu sa svim nitima threads**.**append**(**thread1**)** threads**.**append**(**thread2**)**

# Cekaj dok se sve niti ne izvrse

**for** t **in** threads**:**

t**.**join**()**

**print** "\nIzlazim iz glavne niti\n"

Slika na kojoj se prikazuje tekst

Opis je automatski generiran

**PRIORITETI KOD VIŠENITNOSTI**

*Queue* modul omogućava kreiranje novog queue objekta koji može sadržavati određen broj elemenata. Metode u Queue modulu su:

• **get()** – uklanja i vraća element iz reda čekanja.

• **put()** – dodaje element u red čekanja.

• **qsize()** – vraća broj elemenata koji su trenutno u redu čekanja

• **empty()** – vraća True ukoliko je red čekanja prazan; inače False

• **full()** – vraća True ukoliko je red čekanja pun; inače False

**PRIMJER**

#threading\_primjer\_red\_cekanja.py

#Mrezno programiranje LABno6 2019

**import** Queue **import** threading **import** time

exitFlag **=** 0

**class myThread (**threading**.**Thread**):**

**def** init **(**self**,** threadID**,** name**,** q**):** threading**.**Thread**.** init **(**self**)** self**.**threadID **=** threadID

self**.**name **=** name self**.**q **=** q

**def** run**(**self**):**

**print** "Pokrecem nit " **+** self**.**name

process\_data**(**self**.**name**,** self**.**q**)**

**print** "Izlazim iz niti " **+** self**.**name

**def** process\_data**(**threadName**,** q**):**

**while not** exitFlag**:**

queueLock**.**acquire**()**

**if not** workQueue**.**empty**():** data **=** q**.**get**()** queueLock**.**release**()**

**print** "%s procesuira %s" **% (**threadName**,** data**)**

**else:**

queueLock**.**release**()**

time**.**sleep**(**1**)**

threadList **= [**"Thread-1"**,** "Thread-2"**,** "Thread-3"**]**

nameList **= [**"Jedan"**,** "Dva"**,** "Tri"**,** "Cetiri"**,** "Pet"**]**

queueLock **=** threading**.**Lock**()**

workQueue **=** Queue**.**Queue**(**10**)**

threads **= []**

threadID **=** 1

# Kreiraj nove niti

**for** tName **in** threadList**:**

thread **=** myThread**(**threadID**,** tName**,** workQueue**)**

thread**.**start**()** threads**.**append**(**thread**)** threadID **+=** 1

# Napuni red cekanja queueLock**.**acquire**() for** word **in** nameList**:**

workQueue**.**put**(**word**)**

queueLock**.**release**()**

# Cekaj da se red cekanja isprazni

**while not** workQueue**.**empty**():**

**pass**

# Obavijesti niti da je vrijeme za izlazak exitFlag **=** 1

# Cekaj dok se sve niti ne izvrse

**for** t **in** threads**:**

t**.**join**()**

**print** "\nIzlazim iz glavne niti\n"

Slika na kojoj se prikazuje tekst

Opis je automatski generiran**ZAKLJUČNO**

o Uz sav kod treba dodati datetime modul (ispis trenutnog vremena) i local\_machine\_info podatke

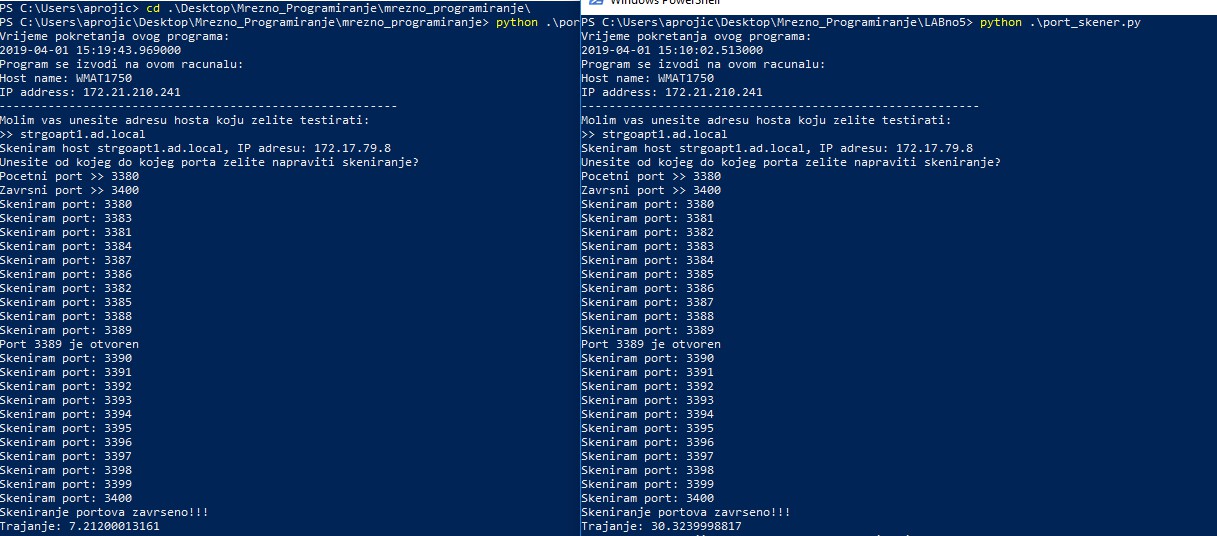
o Treba priložiti screenshotove svih primjera

o Sve treba biti na gitu i poslano na mail

Program iz LABno5 (port skener) prilagodite na način da radi preko threadova. Ukoliko bi išli skenirati veći broj portova na način na koji je to napravljeno u originalnom programu to bi zahtijevalo previše vremena. Možemo iskoristiti niti, te to drastično ubrzati.

Naravno, izračunajte vrijeme izvršenja za vašu aplikaciju iz LABno5 i iz ove laboratorijske vježbe na isti host i priložite screenshot.

Vidi primjer niže:



Slika na kojoj se prikazuje tekst

Opis je automatski generiran

**Savjeti:**

o [https://docs.python.org/2/library/threading.html (](https://docs.python.org/2/library/threading.html)*koristiti threading.Lock()*)

o <https://docs.python.org/2/library/queue.html>

o *Na ova dva linka je sve što trebate znati – pogledajte primjere koda na dnu svake stranice*

o *Ukoliko već niste u LABno5 savjetujem da dio koda koji radi skeniranje pozivate preko funkcije. Biti će vam lakše napraviti ovu vježbu, a i dobra je praksa svaki dio koda koji radi neku specifičnu stvar izdvojiti u funkciju*