**პარალელური და მრავალგანშტოებიანი პროგრამირება**

**ერთბირთვიანი პროცესორი**

განვიხილოთ მაგალითი

import time

def do\_something():

  print("Sleeping 1 second...")

  time.sleep(1)

  return "Done sleeping!"

# ===============

do\_something()

Sleeping 1 second...

დავთვალოთ რეალურად რამდენი ხანი მუშაობს პროგრამა

import time

def do\_something():

print("Sleeping 1 second...")

time.sleep(1)

return "Done sleeping!"

# ===============

start = time.time()

do\_something()

end = time.time()

print(f"Finished at {end - start} seconds...")

Sleeping 1 second...

Finished at 1.0102996826171875 seconds...

ჩავამატოთ do\_something() ფუნქციის კიდევ ერთი გამოძახება

import time

def do\_something():

print("Sleeping 1 second...")

time.sleep(1)

return "Done sleeping!"

# ===============

start = time.time()

do\_something()

do\_something()

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 1 second...

Sleeping 1 second...

Finished at 2.0285518169403076 seconds...

აქ ხდება ასეთი რამ, ჯერ სრულდება do\_something() ფუნქცია. როდესაც მივა time.sleep(1) ბრძანების შესრულებაზე, პროგრამა პაუზდება და პროცესორის ბირთვი არ არის დაკავებული. ამ დროს მეორე do\_something() ფუნქცია არ გამოიძახება. მოკლედ რომ ვთქვათ, სანამ პირველი გამოძახებული ფუნქცია ბოლომდე არ შესრულდება, მეორე არ გაეშვება. ამიტომაც 2 წამზე ოდნავ მეტი სჭირდება პროგრამის შესრულებას. ჩვენ ვერ ვიყენებთ კომპიუტერის რესუსრს და პროცესი ნელა მიმდინარეობს.

კოდი გადავაკეთოთ პროცესის შესაცვლელად... დავაიმპორტოთ threading მოდული, რათა პროცესები უფრო ეფექტურად წარვმართოთ.

import time, threading

def do\_something():

print("Sleeping 1 second...")

time.sleep(1)

return "Done sleeping!"

# ===============

start = time.time()

t1 = threading.Thread(target=do\_something) # პროცესის შექმნა

t1.start() # შექმნილი პროცესის გაშვება

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 1 second...

Finished at 0.0009982585906982422 seconds...

ახლა გვაქს პატარა პრობლემა: წესით პროგრამას 1 წამი უნდა დაეძინა time.sleep(), მაგრამ იბეჭდება Finished in 0.0020008087158203125 seconds... რატომ?

thread-ის შექმნით პროგრამა აღარ ელოდება ფუნქციის დასრულებას და ბეჭდავს Finished in 0.0.0020008087158203125.

Sleeping 1 second! იბეჭდება ფუნქციაში ჩაწერილი კოდის თანმიმდევრობით. შევცვალოთ თანმიმდევრობა და გავუშვათ პროგრამა.

import time, threading

def do\_something():

time.sleep(1)

print("Sleeping 1 second...")

return "Done sleeping!"

# ===============

start = time.time()

t1 = threading.Thread(target=do\_something) # პროცესის შექმნა

t1.start() # შექმნილი პროცესის გაშვება

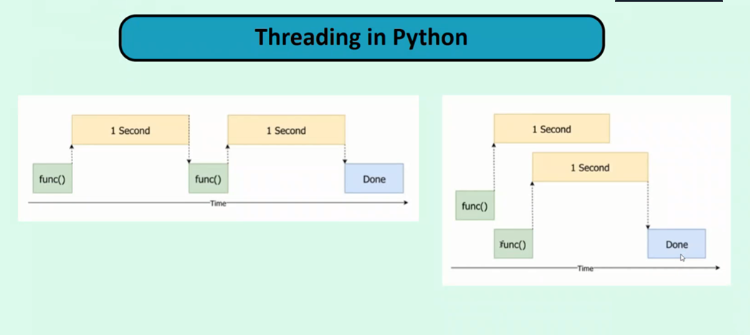
end = time.time()

print(f"Finished at {end - start} seconds...")

Finished at 0.0010018348693847656 seconds...

Sleeping 1 second...

შეიცვალა დაბეჭდვის თანმიმდევრობა. სანამ ფუნქციას „ეძინა“ 1 წამი, მანამდე დაიბეჭდა Finished in 0.001003265380859375 seconds... რაც python-ს არ ახასიათებს. პითონი ზემოდან ქვემოთ ასრულებს ბრძანებებს თანმიმდევრულად; ჩვენ კი შევცვალეთ თანმიმდევრობა, დავყავით პროცესები.



Finished at 0.001003265380859375 seconds... დაბეჭდვა მთლად სწორი არ არის. წესით პროგარამა 1 წამით „იძინებს“... ამიტომ ამ 1 წამს უნდა დაემატოს 0.001003265380859375 წამი. გამოდის, პროგრამა მუშაობს 1.001003265380859375 წამი. ამ შემთხვევაში ბოლო დაბეჭდვა print(f"Finished at {end - start} seconds...") არასწორია. სანამ thread არ მორჩება, არ უნდა დაიბეჭდოს Finished at 0.001003265380859375 seconds... ამისთვის გამოვიყენოთ მეთოდი join(), რათა სწორად დათვალოს პროგრამის მუშაობის დრო.

import time, threading

def do\_something():

print("Sleeping 1 second...")

time.sleep(1)

return "Done sleeping!"

# ===============

start = time.time()

t1 = threading.Thread(target=do\_something)

t1.start()

t1.join()

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 1 second...

Finished at 1.0078034400939941 seconds...

ახალა, შევქმნათ მეორე პროცესი, შემდეგ დავიწყოთ ორივე პროცესი start() და ორივესთვის გამოვიყენოთ join() მეთოდი.

import time, threading

def do\_something():

print("Sleeping 1 second...\n")

time.sleep(1)

return "Done sleeping!"

# ===============

start = time.time()

t1 = threading.Thread(target=do\_something)

t2 = threading.Thread(target=do\_something)

t1.start()

t2.start()

t1.join()

t2.join()

end = time.time()

print(f"Finished at {end - start} seconds...")

Sleeping 1 second...

Sleeping 1 second...

Finished at 1.013627529144287 seconds...

აქ წესით, პროგრამას 2 წამზე მეთასედი წამით მეტი უნდა ემუშავა, მაგრამ ორმა პარალელურმა პროცესმა გაანახევრა მუშაობის დრო.

2 წამის განახევრება დიდი შვება არ არის. რა მოხდება, თუ do\_something() ფუნქცია ათასჯერ, ათი ათასჯერ ან მილიონჯერ არის გაშვებული? მოდით, ხსენებული ფუნქცია გავუშვათ 10-ჯერ და ვნახოთ 10 წამი მოუნდება თუ არა პროგრამა მუშაობას.

import time, threading

def do\_something():

print("Sleeping 1 second...")

time.sleep(1)

return "Done sleeping!"

# ===============

start = time.time()

threads = []

for \_ in range(10):

t = threading.Thread(target=do\_something)

t.start()

threads.append(t)

for t in threads:

t.join()

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second... Sleeping 1 second... Sleeping 1 second...

Finished at 1.022209882736206 seconds...

გადავაკეთოთ do\_something() ფუნქცია. იგი მორგებულია მხოლოდ 1 წამზე. მოვარგოთ მისი მუშაობა რამდენიმე წამზე. ამ ფუნქციას გადავცეთ პარამეტრი seconds.

import time, threading

def do\_something(seconds):

print(f"Sleeping {seconds} second...")

time.sleep(seconds)

return f"Done sleeping in {seconds} seconds..."

# ===============

start = time.time()

threads = []

for \_ in range(10):

t = threading.Thread(target=do\_something, args=(2,))

t.start()

threads.append(t)

for t in threads:

t.join()

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 2 second...

Sleeping 2 second...

Sleeping 2 second...

Sleeping 2 second...

Sleeping 2 second...

Sleeping 2 second...

Sleeping 2 second...

Sleeping 2 second...

Sleeping 2 second...

Sleeping 2 second...

Finished at 2.0288307666778564 seconds...

t = threading.Thread(target=do\_something, args=(2,)) args პარამეტრს გადავცეთ წამების სხვადასხვა რაოდენობა. სხვა სიტყვებით რომ ვთქვათ, do\_something() ფუნქციას წამების სხვადასხვა რაოდენობა გადავცეთ. ყველაზე მარტივია გამოვიყენოთ random მოდულიდან randint() ფუნქცია:

import time, threading

from random import randint

def do\_something(seconds):

print(f"Sleeping {seconds} second...")

time.sleep(seconds)

return f"Done sleeping in {seconds} seconds..."

# ===============

start = time.time()

threads = []

for \_ in range(10):

t = threading.Thread(target=do\_something, args=(randint(1, 3),))

t.start()

threads.append(t)

for t in threads:

t.join()

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 3 second...

Sleeping 1 second...

Sleeping 2 second...

Sleeping 3 second...

Sleeping 1 second...

Sleeping 3 second...

Sleeping 2 second...

Sleeping 3 second...Sleeping 3 second...Sleeping 2 second...

Finished at 3.0255277156829834 seconds...

შეიძლება ითქვას, კოდი „ძველმოდურადაა“ ჩაწერილი (გამოყენებულია import threading მოდული)

ახლა განვიხილოთ პროცესების შექმნის მეორე გაზა.

კოდი გადავაკეთოთ „ახლებურად“: გამოვიყენოთ მოდული concurrent.futures. ამ მოდულის გამოყენებით კოდი უფრო კომპაქტურია.

import time, concurrent.futures

def do\_something(seconds):

print(f"Sleeping {seconds} second...")

time.sleep(seconds)

return f"Done sleeping in {seconds} seconds..."

# ===============

start = time.time()

# threads = []

# for \_ in range(10):

# t = threading.Thread(target=do\_something, args=(randint(1, 3),))

# t.start()

# threads.append(t)

# for t in threads:

# t.join()

with concurrent.futures.ThreadPoolExecutor() as executor:

executor.submit(do\_something, 1)

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 1 second...

Finished at 1.0076820850372314 seconds...

do\_something() ფუნქციიდან დაბრუნებილი ტექსტი return f"Done sleeping in {seconds} seconds..." არსად იბეჭდება. ვცადოთ დაბეჭდვა.

executor.submit(do\_something, 1) შევინახოთ f1 ცვლადში. შემდეგ კი დავბეჭდოთ f1.result(), რომელიც ტერმინალში გამოიტანს do\_something() ფუნქციის მიერ დასაბრუნებელ ინფორმაციას return f"Done Sleeping {seconds}".

import time, concurrent.futures

def do\_something(seconds):

print(f"Sleeping {seconds} second...")

time.sleep(seconds)

return f"Done sleeping in {seconds} seconds..."

# ===============

start = time.time()

with concurrent.futures.ThreadPoolExecutor() as executor:

f1 = executor.submit(do\_something, 1)

print(f1.result())

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 1 second...

Done sleeping in 1 seconds...

Finished at 1.0628595352172852 seconds...

ახლა, შევქმნათ რამდენიმე პროცესი. არგუმენტად ჯერ ისევ 1 წამს გადავცემთ. გავაკეთოთ რამდენიმე ერთწამიან პროცესი.

import time, concurrent.futures

def do\_something(seconds):

print(f"Sleeping {seconds} second...")

time.sleep(seconds)

return f"Done sleeping in {seconds} seconds..."

# ===============

start = time.time()

with concurrent.futures.ThreadPoolExecutor() as executor:

results = [executor.submit(do\_something, 1) for \_ in range(10)]

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 1 second...Sleeping 1 second...Sleeping 1 second...Sleeping 1 second...Sleeping 1 second...Sleeping 1 second...Sleeping 1 second...Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Finished at 1.0154039859771729 seconds...

რეზულტატი Done Sleeping 1, რომელიც ადრე დავბეჭდეთ, აქ არ იბეჭდება. იგი უნდა დაიბეჭდოს ფუნქციის დასრულების შემდეგ. results = [executor.submit(do\_something, 1) for \_ in range(10)] პროცესი როდესაც დასრულდება, მაშინ უნდა დაიბეჭდოს Done Sleeping 1 second... მაგრამ ჩვენ არ ვიცით რომელი პროცესი როდის დასრულდება... შევიტანოთ ცვლილება კოდში.

import time, concurrent.futures

def do\_something(seconds):

print(f"Sleeping {seconds} second...")

time.sleep(seconds)

return f"Done sleeping in {seconds} seconds..."

# ===============

start = time.time()

with concurrent.futures.ThreadPoolExecutor() as executor:

results = [executor.submit(do\_something, 1) for \_ in range(10)]

for f in concurrent.futures.as\_completed(results):

print(f.result())

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 1 second...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Finished at 1.0402824878692627 seconds...

ამ მაგალითში do\_something() ფუნქციას პარამეტრად გადავცემთ 1 წამს. მოდით, ახლა, გადავცეთ წამების სხვადასხვა მნიშვნელობები...

import time, concurrent.futures

def do\_something(seconds):

print(f"Sleeping {seconds} second...")

time.sleep(seconds)

return f"Done sleeping in {seconds} seconds..."

# ===============

start = time.time()

with concurrent.futures.ThreadPoolExecutor() as executor:

secs = [1, 4, 3, 4, 2, 1, 3, 4, 1, 3]

results = [executor.submit(do\_something, sec) for sec in secs]

for f in concurrent.futures.as\_completed(results):

print(f.result())

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 3 second...

Sleeping 1 second...

Sleeping 2 second...

Sleeping 4 second...

Sleeping 4 second...

Sleeping 1 second...

Sleeping 3 second...

Sleeping 4 second...

Sleeping 1 second...

Sleeping 3 second...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 1 seconds...

Done sleeping in 2 seconds...

Done sleeping in 3 seconds...

Done sleeping in 3 seconds...

Done sleeping in 3 seconds...

Done sleeping in 4 seconds...

Done sleeping in 4 seconds...

Done sleeping in 4 seconds...

Finished at 4.040219783782959 seconds...

იგივე ამოცანა გავაკეთოთ map() ფუნქციის გამოყენებით.

results = [executor.submit(do\_something, sec) for sec in secs] მაგივრად გამოვიყენოთ map() ფუნქცია.

ასევე უნდა შევცვალოთ result-ის ბეჭდვა.

import time, concurrent.futures

def do\_something(seconds):

print(f"Sleeping {seconds} second...")

time.sleep(seconds)

return f"Done sleeping in {seconds} seconds..."

# ===============

start = time.time()

with concurrent.futures.ThreadPoolExecutor() as executor:

secs = [1, 4, 3, 4, 2, 1, 3, 4, 1, 3]

# results = [executor.submit(do\_something, sec) for sec in secs]

results = executor.map(do\_something, secs)

for result in results:

print(result)

end = time.time()

print(f"\nFinished at {end - start} seconds...")

Sleeping 4 second...

Sleeping 2 second...

Sleeping 4 second...

Sleeping 1 second...

Sleeping 1 second...

Sleeping 3 second...

Sleeping 3 second...

Sleeping 4 second...

Sleeping 1 second...

Sleeping 3 second...

Done sleeping in 1 seconds...

Done sleeping in 4 seconds...

Done sleeping in 3 seconds...

Done sleeping in 4 seconds...

Done sleeping in 2 seconds...

Done sleeping in 1 seconds...

Done sleeping in 3 seconds...

Done sleeping in 4 seconds...

Done sleeping in 1 seconds...

Done sleeping in 3 seconds...

Finished at 4.04477334022522 seconds...

**ინტერნეტ რესურსიდან ილუსტრაციების ჩამოტვირთვა**

დავწეროთ პროგრამა, რომელიც რომელიმე ვებ-გვერდიდან ჩამოტვირთავს სურათებს და თვალ-ყური ვადევნოთ პროცესებს...

ჯერ მხოლოდ ილუსტრაციები ჩამოვტვირთოთ.

import time, concurrent.futures, requests

img\_urls = [

"http://fotorelax.ru/wp-content/uploads/2016/12/Best-photos-of-the-2016-National-Geographic-16.jpg",

"https://get.wallhere.com/photo/mountains-nature-river-grass-beautiful-landscape-1012301.jpg",

"https://i.pinimg.com/originals/a7/5c/8c/a75c8c8297ecd49090f199435ef2597e.jpg",

"https://i.pinimg.com/originals/33/60/62/3360625518004e8f78125841e8b58115.jpg",

]

t1 = time.time()

for url in img\_urls:

img\_bytes = requests.get(url).content

img\_name = url.split('/')[-1]

with open("images/" + img\_name, "wb") as img\_file: # შექმენნი images საქაღალდე

img\_file.write(img\_bytes)

print(f"{img\_name} was downloaded!")

t2 = time.time()

print(f"\nFinished in {t2 -t1} seconds...")

Best-photos-of-the-2016-National-Geographic-16.jpg was downloaded!

mountains-nature-river-grass-beautiful-landscape-1012301.jpg was downloaded!

a75c8c8297ecd49090f199435ef2597e.jpg was downloaded!

3360625518004e8f78125841e8b58115.jpg was downloaded!

Finished in 28.766044855117798 seconds...

როგორც ვხედავთ სურათების ჩამოტვირთვას 28 წამზე ცოტა მეტი სჭირდება.

ჩვენი ამოცანაა უფრო ნაკლებ დროში ჩამოვტვირთოთ ილუსტრაციები.

ილუტრაციების ჩამოსატვირთად შევქმნათ download() ფუნქცია, რომელსაც არგუმენტად გადავცეთ url მისამართი.

with კონტექსტის საშუალებით გავუშვათ პროცესი.

import time, concurrent.futures, requests

img\_urls = [

"http://fotorelax.ru/wp-content/uploads/2016/12/Best-photos-of-the-2016-National-Geographic-16.jpg",

"https://get.wallhere.com/photo/mountains-nature-river-grass-beautiful-landscape-1012301.jpg",

"https://i.pinimg.com/originals/a7/5c/8c/a75c8c8297ecd49090f199435ef2597e.jpg",

"https://i.pinimg.com/originals/33/60/62/3360625518004e8f78125841e8b58115.jpg",

]

t1 = time.time()

# for url in img\_urls:

# img\_bytes = requests.get(url).content

# img\_name = url.split('/')[-1]

# with open(img\_name, "wb") as img\_file:

# img\_file.write(img\_bytes)

# print(f"{img\_name} was downloaded!")

def download(url):

img\_bytes = requests.get(url).content

img\_name = url.split('/')[-1]

with open("images/" + img\_name, "wb") as img\_file:

img\_file.write(img\_bytes)

print(f"{img\_name} was downloaded!")

with concurrent.futures.ThreadPoolExecutor() as executor:

executor.map(download, img\_urls)

t2 = time.time()

print(f"\nFinished in {t2 -t1} seconds...")

3360625518004e8f78125841e8b58115.jpg was downloaded!

a75c8c8297ecd49090f199435ef2597e.jpg was downloaded!

Best-photos-of-the-2016-National-Geographic-16.jpg was downloaded!

mountains-nature-river-grass-beautiful-landscape-1012301.jpg was downloaded!

Finished in 35.06351447105408 seconds...

**მრავალბირთვიანი პროცესორი**

რასაც აქამდე ვაკეთებდით, პროცესორის მხოლოდ ერთი ბირთვი გამოვიყენეთ (ეფექტურობისთვის).

ახლა ვცადოთ დავალებების (task) სხვადასხვა ბირთვზე განაწილება. განაწილება გულისხმობს, რომ პროცესები პარალელურად, ერთდროულად მიმდინარეობს სხვადასხვა ბირთვში.

import multiprocessing რატომღაც, Windows-ში კოდის გაშვებისას შეცდომას წერს. შესაძლებელია სხვა სისტემაში უშეცდომოდ იმუშაოს.

import multiprocessing, time

def do\_something():

print("Sleeping 1 second!")

time.sleep(1)

return "Done Sleeping..."

# =================

start = time.time()

p1 = multiprocessing.Process(target=do\_something())

p1.start()

p1.join()

end = time.time()

print(f"Time: {end - start} second")

Sleeping 1 second!

Time: 2.0638175010681152 second

ჩავწეროთ if \_\_name\_\_ == '\_\_main\_\_':

import multiprocessing, time

def do\_something():

print("Sleeping 1 second!")

time.sleep(1)

return "Done Sleeping..."

# =================

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

p1 = multiprocessing.Process(target=do\_something)

p1.start()

p1.join()

end = time.time()

print(f"Time: {end - start} second")

Time: 1.0481703281402588 second

შევქმნათ მეორე პროცესი...

import multiprocessing, time

def do\_something():

print("Sleeping 1 second!")

time.sleep(1)

return "Done Sleeping..."

# =================

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

p1 = multiprocessing.Process(target=do\_something)

p2 = multiprocessing.Process(target=do\_something)

p1.start()

p2.start()

p1.join()

p2.join()

end = time.time()

print(f"Time: {end - start} second")

Time: 1.049867868423462 second

შევქმნათ 10 პროცესი, დავიწყოთ ეს პროცესები start()-ით და შევაერთოთ join().

import multiprocessing, time

def do\_something():

print("Sleeping 1 second!")

time.sleep(1)

return "Done Sleeping..."

# =================

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

# p1 = multiprocessing.Process(target=do\_something)

# p2 = multiprocessing.Process(target=do\_something)

# p1.start()

# p2.start()

# p1.join()

# p2.join()

processes = []

for \_ in range(10):

p = multiprocessing.Process(target=do\_something)

p.start()

processes.append(p)

for p in processes:

p.join()

end = time.time()

print(f"Time: {end - start} second")

Time: 1.0992395877838135 second

ახლა, იგივე გავაკეთოთ „თანამეროვე“ ხერხითაც: concurrent.futures მოდულით.

concurrent.futures მოდული განაგებს ორივე პროცესს, როგორც threading, ასევე multiprocessing.

გადავაკეთოთ კოდი ამ მოდულისთვის.

import concurrent.futures, time

def do\_something(seconds):

print(f"Sleeping {seconds} second!")

time.sleep(seconds)

return "Done Sleeping..."

# =================

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

with concurrent.futures.ProcessPoolExecutor() as executor:

p1 = executor.submit(do\_something, 2)

print(p1.result())

end = time.time()

print(f"Time: {end - start} second")

Done Sleeping...

Time: 2.115448236465454 second

შევქმნათ 10 პროცესი...

import concurrent.futures, time

def do\_something(seconds):

print(f"Sleeping {seconds} second!")

time.sleep(seconds)

return "Done Sleeping..."

# =================

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

with concurrent.futures.ProcessPoolExecutor() as executor:

results = [executor.submit(do\_something, 2) for \_ in range(10)]

for p in concurrent.futures.as\_completed(results):

print(p.result())

end = time.time()

print(f"Time: {end - start} second")

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Time: 2.1431233882904053 second

გამოვიყენოთ executor.map() მეთოდი და do\_something() ფუნქციას წამების სხვადსხვა მნიშვნელობები გადავცეთ.

import concurrent.futures, time

def do\_something(seconds):

print(f"Sleeping {seconds} second!")

time.sleep(seconds)

return "Done Sleeping..."

# =================

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

with concurrent.futures.ProcessPoolExecutor() as executor:

secs = (4, 2, 3, 4, 1, 1, 2, 3, 2, 1)

results = executor.map(do\_something, secs)

for result in results:

print(result)

end = time.time()

print(f"Time: {end - start} second")

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Done Sleeping...

Time: 4.133469343185425 second