**PYTHON ТІЛІНДЕ АСИНХРОНДЫ ЖӘНЕ ПАРАЛЛЕЛЬДІ ПРОГРАММАЛАУДЫҢ ТИІМДІЛІГІН ЗЕРТТЕУ**

**Таубай Ернұр Жамбылұлы**

*Әл-Фараби атындағы Қазақ ұлттық университеті,Жасанды интеллектісі бар*

*жоғары жүктелген ақпараттық жүйелер мамандығының 4-курс студенті*

***Аңдатпа****: Бұл ғылыми зерттеуде Python тілінде асинхронды және параллельді программалаудың тиімділігі қарастырылады. Зерттеу Python тіліндегі Global Interpreter Lock (GIL) механизмі мен оның көп ағынды есептеулерге әсерін талдайды. Сондай-ақ, асинхронды программалаудың (asyncio модулі) және параллельді есептеулердің (multiprocessing модулі) ерекшеліктері, олардың I/O операцияларымен жұмыс істеу кезіндегі өнімділігі зерттеледі. Тәжірибелік мысалдар арқылы синхронды, асинхронды және параллельді тәсілдердің жұмыс жылдамдығы мен тиімділігі салыстырылады.*

***Түйін сөздер****: Python, асинхронды программалау, параллелизм, GIL, asyncio, multiprocessing, өнімділік, көп ағынды есептеулер, I/O операциялары, программалау тілдері.*

**ЭФФЕКТИВНОСТЬ АСИНХРОННОГО И ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON.**

***Аннотация****: Бұл ғылыми зерттеуде Python тілінде асинхронды және параллельді программалаудың тиімділігі қарастырылады. Зерттеу Python тіліндегі Global Interpreter Lock (GIL) механизмі мен оның көп ағынды есептеулерге әсерін талдайды. Сондай-ақ, асинхронды программалаудың (asyncio модулі) және параллельді есептеулердің (multiprocessing модулі) ерекшеліктері, олардың I/O операцияларымен жұмыс істеу кезіндегі өнімділігі зерттеледі. Тәжірибелік мысалдар арқылы синхронды, асинхронды және параллельді тәсілдердің жұмыс жылдамдығы мен тиімділігі салыстырылады.*

***Ключевые слова***: *Python, асинхронное программирование, параллелизм, GIL, asyncio, multiprocessing, производительность, многозадачные вычисления, операции ввода-вывода, языки программирования.*

**EXPLORING THE EFFICIENCY OF ASYNCHRONOUS AND PARALLEL PROGRAMMING IN PYTHON.**

***Abstract****: This scientific research examines the efficiency of asynchronous and parallel programming in Python. The study analyzes the Global Interpreter Lock (GIL) mechanism in Python and its impact on multithreaded computations. The characteristics of asynchronous programming (asyncio module) and parallel computations (multiprocessing module) are also explored, focusing on their performance with I/O operations. Practical examples are used to compare the execution speed and efficiency of synchronous, asynchronous, and parallel approaches.*

***Keywords***: *Python, asynchronous programming, parallelism, GIL, asyncio, multiprocessing, performance, multithreaded computations, I/O operations, programming languages*.

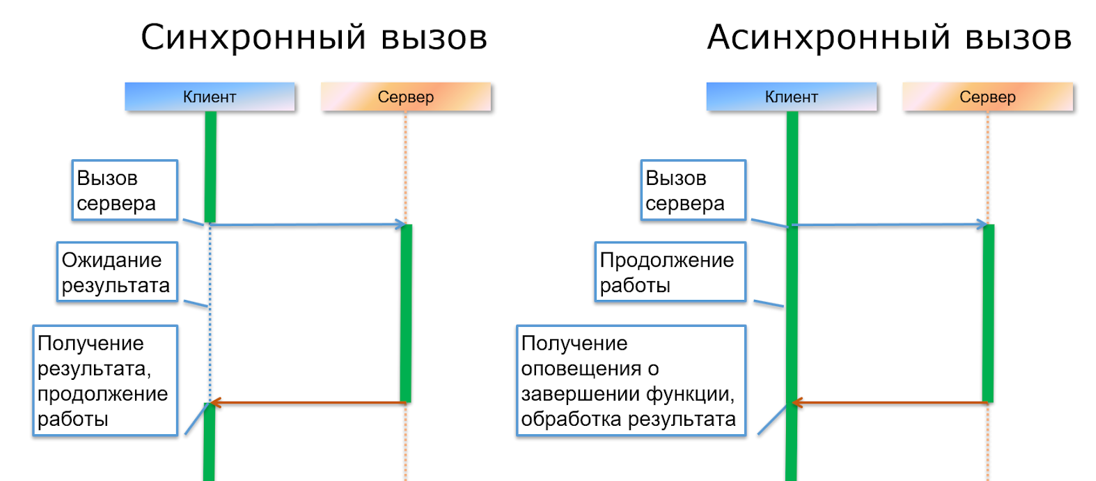
**КІРІСПЕ**

Бағдарламалау тілдерінің арасында Python ерекше орын алады. Оның қарапайымдылығы, әмбебаптығы және ауқымды кітапханаларының арқасында бұл тіл көптеген салада кеңінен қолданылады. Дегенмен, Python-ның танымалдылығына қарамастан, оның ішкі архитектурасы кейбір маңызды шектеулерге ие. Солардың бірі — Global Interpreter Lock (GIL) механизмі. Бұл механизм Python интерпретаторында бір уақытта тек бір ағынның жұмыс істеуіне мүмкіндік береді, нәтижесінде бір ағынды программалау парадигмасы қалыптасқан. Мұндай тәсіл қарапайым сценарийлер үшін жеткілікті болса да, көп тапсырмалы немесе жоғары жүктемелі жүйелерде өнімділікке кедергі келтіруі мүмкін.

Бүгінгі цифрлы әлемде деректердің молдығы және олардың жылдам өңделу қажеттілігі асинхронды тәсілдерге деген сұранысты арттырып отыр. Асинхронды программалау, әсіресе Python тіліндегі asyncio модулі, бұл мәселені шешудің тиімді құралына айналды. Алайда, Python-да асинхронды операцияларды жүзеге асырудың әртүрлі жолдары бар, олардың әрқайсысының өзіндік артықшылықтары мен кемшіліктері бар.

**СИНХРОНДЫ ЖӘНЕ АСИНХРОНДЫ ОПЕРАЦИЯЛАР**

Синхронды операциялар бір тапсырманы аяқтамай тұрып, келесі тапсырманы бастауға мүмкіндік бермейді. Барлық тапсырмалар бірінен соң бірі орындалады. Ал асинхронды операцияларда тапсырмалар бір уақытта басталып, орындалу барысында бірін-бірі бөгемейді.



1-сурет: Синхронды және асинхронды шақырулар

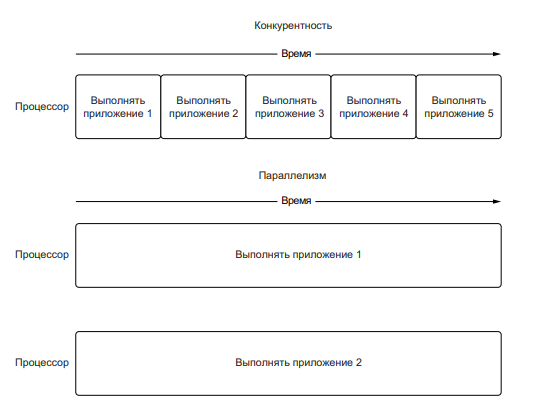
Мысалы, ас үйде бір ас мәзірін дайындап, екінші тағамды пісіріп тұру керек болса, синхронды режимде алдымен бірінші тағамды пісіріп, оның дайын болуын күтіп тұрасыз, сосын ғана екінші тағамды дайындай бастайсыз. Ал асинхронды режимде, бірінші тағам пісіп жатқанда, екінші тағамды дайындауға кірісуге болады. Осылайша, екі тапсырма бір-біріне кедергі келтірмейді.

**КОНКУРЕНТТІЛІК ЖӘНЕ ПАРАЛЛЕЛИЗМ**

Конкуренттілік және параллелизм — есептеу жүйелеріндегі екі түрлі тәсіл, оларды жиі шатастырып жатады, бірақ олардың арасында айтарлықтай айырмашылық бар.

*Конкуренттілік* деп екі немесе одан да көп тапсырмалардың бір уақытта орындалу мүмкіндігін айтады, бірақ олар міндетті түрде бір уақытта орындалмайды. Мысалы, асинхронды программалау тәсілінде бір уақытта бірнеше тапсырма басталып, орындалу барысында бір-бірін бөгемейді. Бірінші тапсырма орындалып жатқанда, екінші тапсырма орындалуға кіріседі, бірақ олар бір уақытта іске қосылмайды, олар өзара уақыттың бөлігімен бөліседі. Бұл тәсіл, әсіресе, бір тапсырманың нәтижесін күтіп тұрған кезде басқа тапсырмаларды орындауға мүмкіндік береді. Осылайша, бір процессор немесе бір ядро арқылы тапсырмаларды кезекпен орындау жүзеге асады, бірақ пайдаланушы үшін тапсырмалар бір уақытта орындалып жатқандай әсер қалдырады.

*Параллелизм* — нақты бір уақытта бірнеше тапсырманың орындалуын білдіреді. Бұл тәсілде бір тапсырманың орындалуы басқа тапсырманың орындалуына кедергі жасамайды, себебі әр тапсырма бөлек процессор немесе процессор ядросы арқылы орындалады. Параллелизм көп ядролы процессорларда тиімді жұмыс істейді, өйткені әрбір ядро өз тапсырмасын параллель түрде орындай алады. Бұл тәсілде барлық тапсырмалар нақты уақыт аралығында орындалады және бұл жұмысты тездетуге мүмкіндік береді, өйткені бірнеше тапсырма бір уақытта өңделеді.



2-сурет: Конкуренттілік және параллелизмнің архитектурасы

**ПОТОКТАР МЕН ПРОЦЕСТЕР**

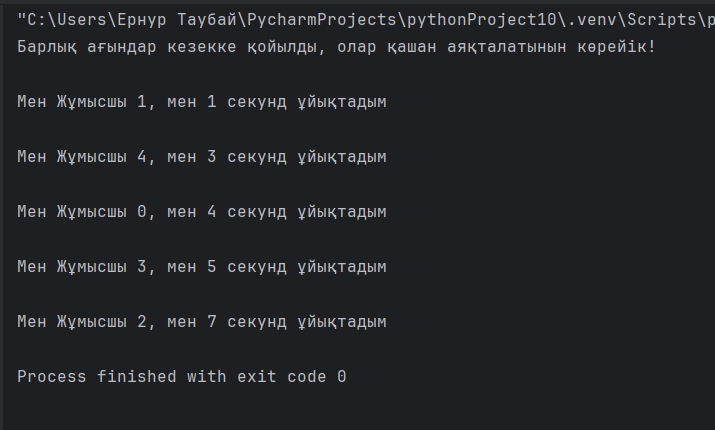
Python тілі ұзақ уақыт бойы потоктарды қолдап келеді. Потоктар — бұл операцияларды бір уақытта орындауға мүмкіндік беретін құрал. Алайда Python-ның Global Interpreter Lock (GIL) механизмі потоктардың нақты параллелизмді қамтамасыз етуіне кедергі келтіреді. GIL тек бір уақытта бір ғана потокты орындай алады, бұл көп ағынды есептеулердің тиімділігін шектейді.

*Потоктар (threads)*

Мысал ретінде келесі кодты қарастырайық. Бұл код бірнеше потокты іске қосады, олар бір уақытта асинхронды түрде жұмыс істейді.

import threading  
import time  
import random  
  
def worker(number):  
 sleep = random.randrange(1, 10)  
 time.sleep(sleep)  
 print(f"\nМен Жұмысшы {number}, мен {sleep} секунд ұйықтадым")  
  
for i in range(5):  
 t = threading.Thread(target=worker, args=(i,))  
 t.start()  
  
print("Барлық ағындар кезекке қойылды, олар қашан аяқталатынын көрейік!")

Нәтиже:



Бұл кодта біз 5 жұмысшыны іске қосып, олар әртүрлі уақытта ұйықтап жатса да, барлық операциялар асинхронды түрде орындалады. Әр жұмысшының ұйықтап тұрған уақытын кездейсоқ таңдаймыз.

*Global interpreter lock (gil):*

*GIL* — бұл CPython интерпретаторында орындалатын бір уақытта тек бір ағынды басқаруға мүмкіндік беретін механизм. GIL арқасында бірнеше поток бір уақытта орындалмайды. Алайда, I/O операциялары үшін бұл кедергі болмайды, себебі GIL I/O операцияларын орындауға кедергі келтірмейді.

*Процестер (processes)*

Егер параллелизм қажет болса, Python multiprocessing модулін ұсынады. Бұл модуль бірнеше процесті іске қосуға мүмкіндік береді, бұл бірнеше ядроның жұмысын тиімді пайдалану үшін қажет.

import multiprocessing  
import time  
import random  
  
def worker(number):  
 sleep = random.randrange(1, 10)  
 time.sleep(sleep)  
 print(f"\nМен Жұмысшы {number}, мен {sleep} секунд ұйықтадым")  
if \_\_name\_\_ == '\_\_main\_\_':  
 for i in range(5):  
 t = multiprocessing.Process(target=worker, args=(i,))  
 t.start()  
  
 print("Барлық ағындар кезекке қойылды, олар қашан аяқталатынын көрейік!")

Бұл кодта біз әр процесс үшін бөлек ядро пайдаланамыз. Бұл параллелизмнің нақты мысалы, себебі әр процесс өз процессор уақытын пайдаланады.

**ASYNCIO — ЖАҢА МҮМКІНДІКТЕР**

Асинхронды программалау — бұл бір уақытта бірнеше тапсырманы орындауға мүмкіндік беретін тәсіл. Бұл тәсіл жұмыс істеп жатқан ағынды бөгемей, I/O операцияларын немесе басқа блокталатын тапсырмаларды тиімді орындауға көмектеседі. Асинхронды программалаудың негізгі принципі — блокталмайтын операцияларды пайдалану, яғни бір операция аяқталмай тұрып, келесі операцияны орындауға мүмкіндік беру.

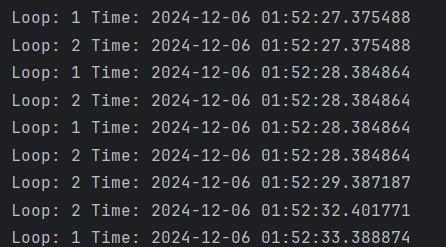
Асинхронды тәсілдің ең басты ерекшелігі — **event loop** (оқиғалар циклі). Бұл цикл қолданушыдан немесе жүйеден келетін оқиғаларды үнемі бақылап, олардың орындалуын қамтамасыз етеді. Іс жүзінде, event loop бір ағынды ортаны пайдалана отырып, көптеген тапсырмаларды орындай алады. Бір тапсырма аяқталмай тұрып, event loop келесі тапсырманы орындауға кіріседі, бірақ бірінші тапсырма орындалған кезде ғана оның нәтижесін алады. Бұл тәсіл блокталуды болдырмайды және жоғары өнімділікті қамтамасыз етеді.

Python тілінде асинхронды программалау үшін арнайы модуль — asyncio бар. asyncio — бұл асинхронды код жазуға арналған құралдар жиынтығы, ол event loop, асинхронды операцияларды орындау және күтуді жеңілдету үшін көптеген абстракциялар ұсынады. Бұл модуль асинхронды функцияларды, мысалы, async және await синтаксисін пайдалана отырып, іске қосуға мүмкіндік береді. asyncio арқылы кодты блоктамай параллель түрде орындауға болады, осылайша I/O операциялары немесе сыртқы жүйелермен жұмыс істеу кезінде көп уақытты үнемдеуге болады.

Асинхронды енгізу-шығару операциялары үшін asyncio модулі әсіресе пайдалы. Бұл модуль циклдік оқиғаларды (event loop) қолдану арқылы бір уақытта бірнеше операцияны орындауға мүмкіндік береді, бірақ ол I/O операцияларын блоктамайды.

import asyncio  
import datetime  
import random  
  
async def my\_sleep\_func():  
 await asyncio.sleep(random.randint(0, 5))  
  
async def display\_date(num, loop):  
 end\_time = loop.time() + 50.0  
 while True:  
 print(f"Loop: {num} Time: {datetime.datetime.now()}")  
 if (loop.time() + 1.0) >= end\_time:  
 break  
 await my\_sleep\_func()  
  
loop = asyncio.get\_event\_loop()  
asyncio.ensure\_future(display\_date(1, loop))  
asyncio.ensure\_future(display\_date(2, loop))  
loop.run\_forever()

Нәтиже:



Бұл кодта біз бірнеше асинхронды тапсырмаларды орындап, оларды күтеміз. asyncio барлық асинхронды функциялардың орындалуын басқаруға көмектеседі, бұл әсіресе көп I/O операциялары кезінде тиімді.

**ҚОРЫТЫНДЫ**

Python-да асинхронды және параллелизмді қолдану тиімділігі бағдарламаның талаптарына байланысты. *Синхронды және асинхронды**операциялар* арасындағы айырмашылық, асинхронды режимде бірнеше тапсырманы бір уақытта орындауға мүмкіндік береді, бұл әсіресе *I/O операцияларында* тиімді. *Конкуренттілік* бірнеше тапсырманы кезекпен орындауды білдірсе, *параллелизм* нақты уақытта бірнеше тапсырманы орындауға мүмкіндік береді.

*Потоктар мен**процестер* Python-да көп ағынды есептеулерде пайдалы, бірақ *Global Interpreter Lock (GIL)* параллелизмді шектейді, сондықтан *multiprocessing* модулі көп есептеулер үшін тиімді. *Asyncio* асинхронды енгізу-шығару үшін оңтайлы, себебі ол I/O операцияларын блоктамайды.

Қай әдісті таңдау, қолданбаңыздың I/O немесе есептеу жұмыстарына байланысты өзгеріп, әр әдіс өз жағдайында тиімді жұмыс істейді.

**ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ**

1. Мэттью Фаулер. Asyncio и конкурентное программирование на Python. Нью-Юорк: Manning Publications. 2022.
2. Күшейтілген оқыту. URL: https://docs.python.org/3/library/asyncio.html (жүгінген күні: 06.12.2024).
3. Ян Палах. Параллельді бағдарламалау Python тілінде, Бирмингем: Packt Publishing. 2014.
4. Күшейтілген оқыту. URL: https://en.wikipedia.org/wiki/Asynchronous\_I/O(жүгінген күні: 06.12.2024).