دانشگاه لرستان

به نام خدا

تمرین درس پردازش موازی

دانشجو:

پریا تاری

شماره دانشجویی:

40211415012

استاد مربوطه:

دکتر آرمین رشنو

تیر 1403

Thread-Based Parallelism

1) Defining a thread

سناریو اول:

```
6   app = FastAPI()
7
8   output_messages = []
9
    4 usages  new *
10  def my_func(thread_number: int):
11      message = f"my_func called by thread N°{thread_number}"
12      output_messages.append(message)
13
    3 usages  new *
14  class ThreadCount(BaseModel):
15      thread_count: int
    new *
16  @app.post("/body1/")
17  async def body_scenario(thread_count: ThreadCount):
18      global output_messages
19      output_messages = []
20      threads = []
21      for i in range(thread_count.thread_count):
22          thread = threading.Thread(target=my_func, args=(i,))
23          threads.append(thread)
24          thread.start()
25      for thread in threads:
26          thread.join()
27      return output_messages
28
```

ابتدا یک نمونه جدید از برنامه FastAPI را ایجاد می‌کنیم.بعد یک لیست عمومی به نام `output_messages` ایجاد می‌کنیم تا پیام‌های تولید شده توسط نخ‌ها را ذخیره کنیم.

تابع `my_func` یک پارامتر `thread_number` (شماره نخ) را می‌پذیرد و در داخل تابع، یک پیام ایجاد می‌شود که نشان می‌دهد این تابع توسط نخ شماره خاصی فراخوانی شده است سپس این پیام به لیست `output_messages` اضافه می‌شود.

خط 14 یک مدل داده‌ای به نام `ThreadCount` را از کلاس `BaseModel` تعریف می‌کند که شامل یک فیلد `thread_count` از نوع عدد صحیح (int)است.

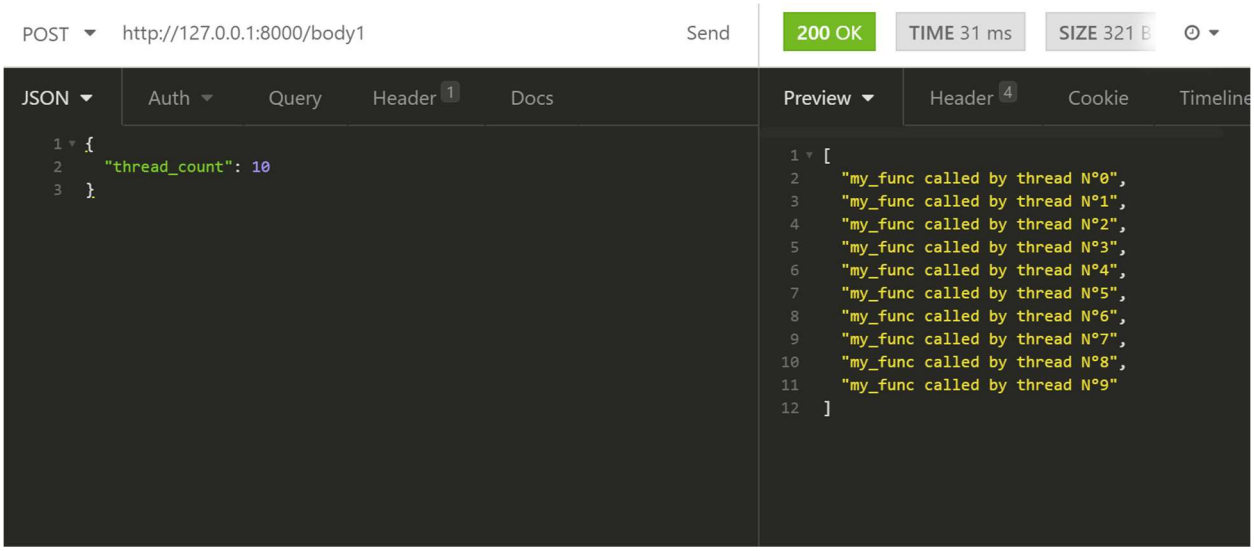
خط 16 و 17 یک مسیر HTTP POST به نام `/body1/`تعریف می‌کند که تابع `body_scenario` را فراخوانی می‌کند و تابع یک پارامتر به نام `thread_count` می‌پذیرد که از نوع مدل `ThreadCount` است.

خط 18 متغیر `output_messages` را به صورت عمومی اعلام می‌کند تا بتواند در تابع استفاده شود و لیست `output_messages` را خالی می‌کند تا مطمئن شود که هیچ پیام قبلی در آن نیست بعد یک لیست خالی به نام `threads`ایجاد می‌کند تا نخ‌های جدید را در آن ذخیره کند.

حلقه به تعداد `thread_count.thread_count` تکرار می‌شود، در هر تکرار، یک نخ جدید ایجاد می‌شود که تابع `my_func` را با شماره نخ فعلی (`i`) به عنوان آرگومان فراخوانی می‌کند، نخ ایجاد شده به لیست `threads`اضافه می‌شود و بعد نخ شروع به کار می‌کند .

این حلقه بر روی تمام نخ‌های ایجاد شده در لیست `threads`تکرار می‌شود و `thread.join()` باعث می‌شود که برنامه منتظر بماند تا نخ فعلی کار خود را به پایان برساند و در نهایت خط 27 لیست `output_messages` را که شامل پیام‌های تولید شده توسط نخ‌ها است، برمی‌گرداند.

http://localhost:7000/body1



سناریو دوم:

```python
new *
@app.post("/body2/")
async def body_scenario(thread_count: ThreadCount):
    global output_messages
    output_messages = []
    threads = []
    for i in range(thread_count.thread_count):
        thread = threading.Thread(target=my_func, args=((thread_count.thread_count - 1) - i,))
        threads.append(thread)
        thread.start()
        thread.join()

    return output_messages
```

POST http://127.0.0.1:8000/body2 Send 200 OK TIME 15 ms SIZE 321 B

JSON Auth Query Header 1 Docs

```
{
  "thread_count": 10
}
```

Preview Header 4 Cookie Timeline

```
[
  "my_func called by thread N°9",
  "my_func called by thread N°8",
  "my_func called by thread N°7",
  "my_func called by thread N°6",
  "my_func called by thread N°5",
  "my_func called by thread N°4",
  "my_func called by thread N°3",
  "my_func called by thread N°2",
  "my_func called by thread N°1",
  "my_func called by thread N°0"
]
```

دراین نسخه، نخ‌ها به ترتیب معکوس شماره‌گذاری شده‌اند (از شماره بالا به پایین) به جای ترتیب مستقیم و `thread.join()` بلافاصله پس از `thread.start()` فراخوانی می‌شود. این باعث می‌شود که هر نخ به ترتیب ایجاد و تکمیل شود قبل از شروع نخ بعدی، در حالی که در نسخه قبلی، همه نخ‌ها به صورت موازی شروع می‌شدند و سپس منتظر تکمیل شدن همه نخ‌ها می‌شدند.

سناریو سوم:

```python
@app.post("/body3/")
async def body_scenario(thread_count: ThreadCount):
    global output_messages
    output_messages = []
    threads = []

    half_count = thread_count.thread_count // 2

    for i in range(half_count, thread_count.thread_count):
        thread = threading.Thread(target=my_func, args=(i,))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()

    threads = []
    for i in range(half_count):
        thread = threading.Thread(target=my_func, args=(i,))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()

    return output_messages
```

POST http://127.0.0.1:8000/body3 Send 200 OK TIME 16 ms SIZE 321 B

JSON Auth Query Header 1 Docs

```
{
  "thread_count": 10
}
```

Preview Header 4 Cookie Timeline

```
[
  "my_func called by thread N°5",
  "my_func called by thread N°6",
  "my_func called by thread N°7",
  "my_func called by thread N°8",
  "my_func called by thread N°9",
  "my_func called by thread N°0",
  "my_func called by thread N°1",
  "my_func called by thread N°2",
  "my_func called by thread N°3",
  "my_func called by thread N°4"
]
```

د ر این نسخه، تعداد نخ‌ها به دو نیم تقسیم می‌شود. ابتدا، نخ‌هایی که شماره آن‌ها بین `half_count` و `thread_count.thread_count` است، ایجاد و اجرا می‌شوند. سپس، نخ‌هایی که شماره آن‌ها بین 0 و `half_count` است، ایجاد و اجرا می‌شوند.

هر مجموعه از نخ‌ها به صورت موازی اجرا می‌شوند و سپس منتظر تکمیل شدن آن‌ها می‌ماند تا مجموعه بعدی شروع شود.

2) Determining the current thread

سناریو اول:

```python
69  def function_A():
70      global output_messages
71      output_messages.append("function_A--> starting")
72      time.sleep(1)
73      output_messages.append("function_A--> exiting")
74
    3 usages  new *
75  def function_B():
76      global output_messages
77      output_messages.append("function_B--> starting")
78      time.sleep(3)
79      output_messages.append("function_B--> exiting")
80
    3 usages  new *
81  def function_C():
82      global output_messages
83      output_messages.append("function_C--> starting")
84      time.sleep(2)
85      output_messages.append("function_C--> exiting")
86
```

تابع `function_A` تعریف شده است. این تابع دو پیام به لیست عمومی `output_messages` اضافه می‌کند:

`"function_A --> starting"` پیامی که نشان دهنده شروع اجرای تابع A است.

`"function_A --> exiting"` پیامی که نشان دهنده پایان اجرای تابع A است

سپس تابع برای 1 ثانیه متوقف می‌شود. دو تابع دیگر هم به همین صورت.

```python
88  @app.get("/path1/{thread_count}")
89  async def path_thread(thread_count: int = Path( default: ..., title="Number of Threads", ge=1, le=3)):
90      global output_messages
91      output_messages = []
92      threads = []
93      functions = [function_A, function_B, function_C]
94      for i in range(thread_count):
95          thread = threading.Thread(target=functions[i])
96          threads.append(thread)
97          thread.start()
98      for thread in threads:
99          thread.join()
100     return output_messages
```

این خط تابع `path_thread` را به صورت ناهمزمان (async) تعریف می‌کند که یک پارامتر به نام `thread_count` می‌پذیرد. پارامتر `thread_count` از نوع عدد صحیح (int) است و باید بین 1 و 3 باشد که به وسیله `ge=1` و `le=3`محدود شده است.

خط 93 یک لیست به نام `functions` ایجاد می‌کند که شامل سه تابع `function_A`، `function_B` و `function_C`است.

این حلقه به تعداد `thread_count` تکرار می‌شود، در هر تکرار، یک نخ جدید ایجاد می‌شود که تابع مشخصی از لیست `functions` را فراخوانی می‌کند بر اساس مقدار `i` ، نخ ایجاد شده به لیست `threads` اضافه می‌شود و نخ شروع به کار می‌کند.

http://localhost:7000/path1/3

GET http://127.0.0.1:8000/path1/3 Send 200 OK TIME 3 s SIZE 148

JSON Auth Query Header 1 Docs Preview Header 4 Cookie Tim

```
1 ...
```

```
1 [
2   "function_A--> starting",
3   "function_B--> starting",
4   "function_C--> starting",
5   "function_A--> exiting",
6   "function_C--> exiting",
7   "function_B--> exiting"
8 ]
```

سناریو دوم:

```
new *
@app.get("/path2/{thread_count}")
async def path_thread(thread_count: int = Path( default: ..., title="Number of Threads", ge=1, le=3)):
    global output_messages
    output_messages = []
    threads = []
    functions = [function_A, function_B, function_C]
    for i in range(thread_count):
        thread = threading.Thread(target=functions[i])
        threads.append(thread)
        thread.start()
        thread.join()

    return output_messages
```

GET http://127.0.0.1:8000/path2/3 Send 200 OK TIME 6.05 s SIZE

JSON Auth Query Header 1 Docs Preview Header 4 Cookie Tim

```
1 ...
```

```
1 [
2   "function_A--> starting",
3   "function_A--> exiting",
4   "function_B--> starting",
5   "function_B--> exiting",
6   "function_C--> starting",
7   "function_C--> exiting"
8 ]
```

سناریو سوم:

```python
@app.get("/path3/{thread_count}")
async def path_thread(thread_count: int = Path(default: ..., title="Number of Threads", ge=1, le=3)):
    global output_messages
    output_messages = []
    threads = []

    thread_A = threading.Thread(target=function_A)
    thread_B = threading.Thread(target=function_B)
    thread_C = threading.Thread(target=function_C)

    threads.append(thread_C)
    thread_C.start()
    thread_C.join()
    threads.append(thread_B)
    thread_B.start()
    thread_B.join()
    threads.append(thread_A)
    thread_A.start()
    thread_A.join()

    return output_messages
```

GET http://127.0.0.1:8000/path3/3 Send 200 OK TIME 6.03 s SIZE

JSON Auth Query Header 1 Docs Preview Header 4 Cookie Tim

```
1 ...
```

```
[
  "function_C--> starting",
  "function_C--> exiting",
  "function_B--> starting",
  "function_B--> exiting",
  "function_A--> starting",
  "function_A--> exiting"
]
```

نخ‌ها به ترتیب `thread_C`، `thread_B` و `thread_A` شروع به کار می‌کنند و هر کدام به ترتیب منتظر تکمیل شدن نخ قبلی می‌مانند.

3) Defining a thread subclass

سناریو اول:

```
143 class MyThread(threading.Thread):
        new *
144     def __init__(self, thread_id, sleep_time):
145         super().__init__()
146         self.thread_id = thread_id
147         self.sleep_time = sleep_time
148
        new *
149     def run(self):
150         global output_messages
151         output_messages.append(f"---> Thread#{self.thread_id} running, belonging to process ID {threading.current_thread().ident}")
152         time.sleep(self.sleep_time)
153         output_messages.append(f"---> Thread#{self.thread_id} over")
154
    1 usage  new *
155 class ThreadInput(BaseModel):
156     thread_id: int
157     sleep_time: float
158
```

خط 143 یک کلاس جدید به نام `MyThread` ایجاد می‌کند که از کلاس `threading.Thread` ارث‌بری می‌کند.

متد سازنده (`__init__`) برای کلاس `MyThread` است. `thread_id` و `sleep_time` را به عنوان پارامتر می‌پذیرد و آن‌ها را به خصوصیات کلاس تخصیص می‌دهد.

متد `run` که هنگام شروع نخ اجرا می‌شود. یک پیام به `output_messages` اضافه می‌کند که نشان می‌دهد نخ با `thread_id` مشخص شروع به اجرا کرده و همچنین شناسه (ID) نخ فعلی را نمایش می‌دهد. بعد نخ را به مدت `sleep_time` متوقف می‌کند. یک پیام دیگر به `output_messages` اضافه می‌کند که نشان می‌دهد نخ با `thread_id` مشخص پایان یافته است.

این کلاس `ThreadInput` از `BaseModel` ارث‌بری می‌کند. دو فیلد `thread_id` از نوع عدد صحیح (`int`) و `sleep_time` از نوع عدد اعشاری (`float`) را تعریف می‌کند.

```
159 @app.post("/body4/")
160 async def run_threads(thread_inputs: list[ThreadInput] = Body(...)):
161     threads = []
162     global output_messages
163     output_messages = []
164
165     start_time = time.time()
166
167     for thread_input in thread_inputs:
168         thread = MyThread(thread_input.thread_id, thread_input.sleep_time)
169         thread.start()
170         threads.append(thread)
171
172     for thread in threads:
173         thread.join()
174
175     end_time = time.time()
176     total_time = end_time - start_time
177
178     output_messages.append("End")
179     output_messages.append(f"--- {total_time} seconds ---")
180
181     return output_messages
182
```

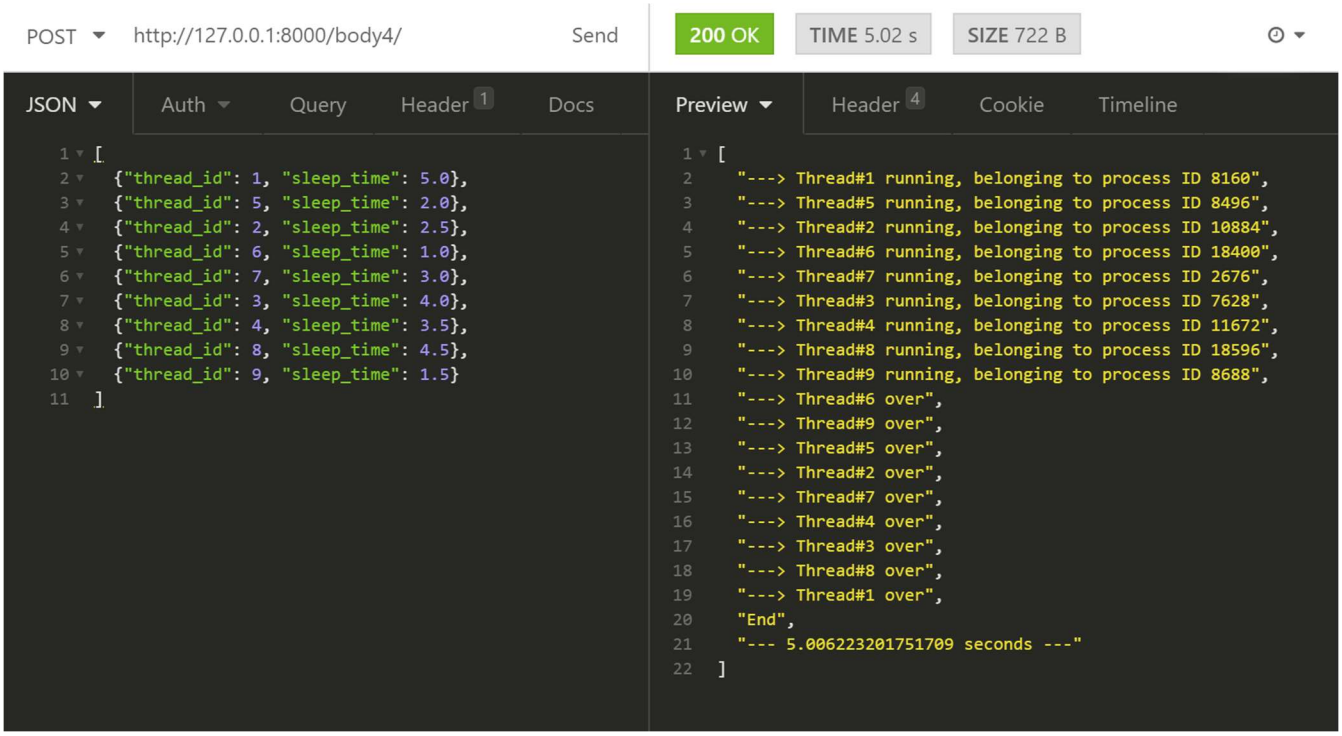
این تابع به صورت ناهمزمان (`async`) است و یک لیست از اشیاء `ThreadInput` به عنوان ورودی می‌پذیرد که از بدنه درخواست (`Body(...)`) گرفته می‌شود. خط 165 زمان شروع را ثبت می‌کند.

حلقه بر روی هر ورودی در `thread_inputs` تکرار می‌شود. یک نخ جدید از نوع `MyThread` ایجاد می‌کند با استفاده از `thread_id` و `sleep_time` ورودی ها را ارسال میکند. بعد نخ ایجاد شده را شروع می‌کند و به لیست `threads` اضافه می‌کند.

حلقه خط 172 بر روی تمام نخ‌های ایجاد شده در لیست `threads` تکرار می‌شود و `thread.join()` باعث می‌شود که برنامه منتظر بماند تا نخ فعلی کار خود را به پایان برساند.

خط 175 زمان پایان را ثبت می‌کند و مدت زمان اجرای کل نخ‌ها را محاسبه می‌کند.

http://localhost:7000/body4/



سناریو دوم:

```python
@app.post("/body5/")
async def run_threads(thread_inputs: list[ThreadInput] = Body(...)):
    threads = []
    global output_messages
    output_messages = []

    start_time = time.time()

    for thread_input in thread_inputs:
        thread = MyThread(thread_input.thread_id, thread_input.sleep_time)
        thread.start()
        threads.append(thread)
        thread.join()

    end_time = time.time()
    total_time = end_time - start_time

    output_messages.append("End")
    output_messages.append(f"--- {total_time} seconds ---")

    return output_messages

```

POST http://127.0.0.1:8000/body5/ Send

200 OK | TIME 27.1 s | SIZE 727 B

JSON | Auth | Query | Header 1 | Docs

```
[
  {"thread_id": 1, "sleep_time": 5.0},
  {"thread_id": 5, "sleep_time": 2.0},
  {"thread_id": 2, "sleep_time": 2.5},
  {"thread_id": 6, "sleep_time": 1.0},
  {"thread_id": 7, "sleep_time": 3.0},
  {"thread_id": 3, "sleep_time": 4.0},
  {"thread_id": 4, "sleep_time": 3.5},
  {"thread_id": 8, "sleep_time": 4.5},
  {"thread_id": 9, "sleep_time": 1.5}
]
```

Preview | Header 4 | Cookie | Timeline

```
[
  "---> Thread#1 running, belonging to process ID 15376",
  "---> Thread#1 over",
  "---> Thread#5 running, belonging to process ID 6816",
  "---> Thread#5 over",
  "---> Thread#2 running, belonging to process ID 12868",
  "---> Thread#2 over",
  "---> Thread#6 running, belonging to process ID 19836",
  "---> Thread#6 over",
  "---> Thread#7 running, belonging to process ID 11132",
  "---> Thread#7 over",
  "---> Thread#3 running, belonging to process ID 19544",
  "---> Thread#3 over",
  "---> Thread#4 running, belonging to process ID 10744",
  "---> Thread#4 over",
  "---> Thread#8 running, belonging to process ID 18348",
  "---> Thread#8 over",
  "---> Thread#9 running, belonging to process ID 11052",
  "---> Thread#9 over",
  "End",
  "--- 27.056869983673096 seconds ---"
]
```

سناریو سوم:

```python
@app.post("/body6/")
async def run_threads(thread_inputs: list[ThreadInput] = Body(...)):
    threads = []
    global output_messages
    output_messages = []

    start_time = time.time()

    for thread_input in thread_inputs:
        if thread_input.thread_id <= 5:
            thread = MyThread(thread_input.thread_id, thread_input.sleep_time)
            thread.start()
            threads.append(thread)
    for thread in threads:
        thread.join()

    for thread_input in thread_inputs:
        if thread_input.thread_id > 5:
            thread = MyThread(thread_input.thread_id, thread_input.sleep_time)
            thread.start()
            threads.append(thread)
    for thread in threads:
        thread.join()

    end_time = time.time()
    total_time = end_time - start_time

    output_messages.append("End")
    output_messages.append(f"--- {total_time} seconds ---")

    return {"output": output_messages}
```

POST http://127.0.0.1:8000/body6/ Send

200 OK TIME 9.53 s SIZE 734 B

JSON Auth Query Header 1 Docs

```json
[
  {"thread_id": 1, "sleep_time": 5.0},
  {"thread_id": 5, "sleep_time": 2.0},
  {"thread_id": 2, "sleep_time": 2.5},
  {"thread_id": 6, "sleep_time": 1.0},
  {"thread_id": 7, "sleep_time": 3.0},
  {"thread_id": 3, "sleep_time": 4.0},
  {"thread_id": 4, "sleep_time": 3.5},
  {"thread_id": 8, "sleep_time": 4.5},
  {"thread_id": 9, "sleep_time": 1.5}
]
```

Preview Header 4 Cookie Timeline

```json
{
  "output": [
    "---> Thread#1 running, belonging to process ID 20292",
    "---> Thread#5 running, belonging to process ID 18692",
    "---> Thread#2 running, belonging to process ID 6232",
    "---> Thread#3 running, belonging to process ID 13072",
    "---> Thread#4 running, belonging to process ID 4928",
    "---> Thread#5 over",
    "---> Thread#2 over",
    "---> Thread#4 over",
    "---> Thread#3 over",
    "---> Thread#1 over",
    "---> Thread#6 running, belonging to process ID 9240",
    "---> Thread#7 running, belonging to process ID 15496",
    "---> Thread#8 running, belonging to process ID 1188",
    "---> Thread#9 running, belonging to process ID 12700",
    "---> Thread#6 over",
    "---> Thread#9 over",
    "---> Thread#7 over",
    "---> Thread#8 over",
    "End",
    "--- 9.524458885192871 seconds ---"
  ]
}
```

4) Thread synchronization with a lock

سناریو اول:

```python
lock = threading.Lock()

class MyThread_2(threading.Thread):
    def __init__(self, thread_id):
        super().__init__()
        self.thread_id = thread_id

    def run(self):
        global output_messages
        global lock

        lock.acquire()
        try:
            output_messages.append(
                f"---> Thread#{self.thread_id} running, belonging to process ID {threading.current_thread().ident}")
        finally:
            lock.release()

        time.sleep(1)

        lock.acquire()
        try:
            output_messages.append(f"---> Thread#{self.thread_id} over")
        finally:
            lock.release()
```

از `with lock:` استفاده می‌کند تا بخش کد داخل آن تنها زمانی اجرا شود که قفل (lock) گرفته شده باشد.

پیامی به `output_messages` اضافه می‌کند که نشان‌دهنده شروع نخ با `thread_id` مشخص و شناسه (ID) نخ فعلی است.

نخ را به مدت 1 ثانیه متوقف می‌کند و دوباره از `with lock:` استفاده می‌کند تا بخش کد داخل آن تنها زمانی اجرا شود که قفل (lock) گرفته شده باشد.

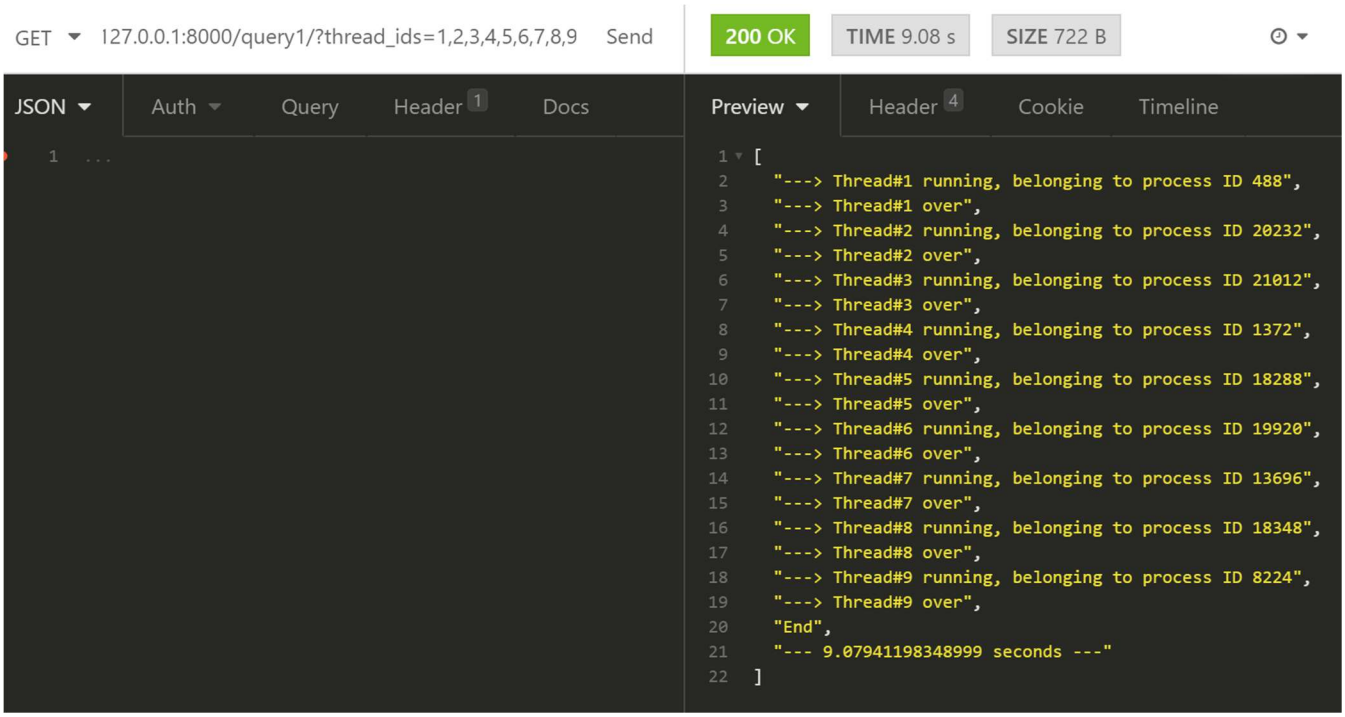
پیامی به `output_messages`اضافه می‌کند که نشان‌دهنده پایان نخ با `thread_id` مشخص است.

```python
@app.get("/query1/")
async def run_threads():
    global output_messages
    output_messages = []
    threads = []

    start_time = time.time()

    for i in range(1, 10):
        thread = MyThread_2(i)
        thread.start()
        threads.append(thread)
        thread.join()

    end_time = time.time()
    total_time = end_time - start_time

    output_messages.append("End")
    output_messages.append(f"--- {total_time} seconds ---")

    return output_messages
```

حلقه از 1 تا 9 تکرار می‌شود، برای هر مقدار `i`، یک نخ جدید از نوع `MyThread_2` ایجاد می‌کند با استفاده از `i` به عنوان `thread_id`.

نخ ایجاد شده را شروع می‌کند و به لیست `threads` اضافه میکند سپس منتظر می‌ماند تا نخ کار خود را به پایان برساند.

http://localhost:7000/query1/?thread_ids=1,2,3,4,5,6,7,8,9



سناریو دوم:

```python
@app.get("/query2/")
async def run_threads():
    global output_messages
    output_messages = []
    threads = []

    start_time = time.time()

    for i in range(1, 10):
        thread = MyThread_2(i)
        thread.start()
        threads.append(thread)
    for thread in threads:
        thread.join()

    end_time = time.time()
    total_time = end_time - start_time

    output_messages.append("End")
    output_messages.append(f"--- {total_time} seconds ---")

    return output_messages
```

GET 127.0.0.1:8000/query2/?thread_ids=1,2,3,4,5,6,7,8,9 Send

200 OK TIME 1.03 s SIZE 725 B

JSON Auth Query Header 1 Docs

```
1 ...
```

Preview Header 4 Cookie Timeline

```
1  [
2    "---> Thread#1 running, belonging to process ID 12424",
3    "---> Thread#2 running, belonging to process ID 11052",
4    "---> Thread#3 running, belonging to process ID 1964",
5    "---> Thread#4 running, belonging to process ID 20268",
6    "---> Thread#5 running, belonging to process ID 8368",
7    "---> Thread#6 running, belonging to process ID 15868",
8    "---> Thread#7 running, belonging to process ID 8404",
9    "---> Thread#8 running, belonging to process ID 15216",
10   "---> Thread#9 running, belonging to process ID 12732",
11   "---> Thread#8 over",
12   "---> Thread#7 over",
13   "---> Thread#6 over",
14   "---> Thread#5 over",
15   "---> Thread#2 over",
16   "---> Thread#1 over",
17   "---> Thread#9 over",
18   "---> Thread#4 over",
19   "---> Thread#3 over",
20   "End",
21   "--- 1.0104260444641113 seconds ---"
22 ]
```

سناریو سوم:

```python
@app.get("/query3/")
async def run_threads():
    global output_messages
    output_messages = []
    threads = []

    start_time = time.time()

    for i in range(1, 10):
        thread = MyThread_2(10 - i)
        thread.start()
        threads.append(thread)
        thread.join()

    end_time = time.time()
    total_time = end_time - start_time

    output_messages.append("End")
    output_messages.append(f"--- {total_time} seconds ---")

    return output_messages
```

در هر تکرار یک نخ جدید از نوع `MyThread_2` با شناسه نخ (`thread_id`) معکوس (از 9 به 1) ایجاد می‌کند.

`10 - i` باعث می‌شود که اولین نخ ایجاد شده `thread_id` برابر با 9 باشد و آخرین نخ ایجاد شده `thread_id` برابر با 1 باشد.

GET 127.0.0.1:8000/query3/?thread_ids=1,2,3,4,5,6,7,8,9 Send

200 OK | TIME 9.09 s | SIZE 724 B

JSON | Auth | Query | Header 1 | Docs

Preview | Header 4 | Cookie | Timeline

```
[
  "---> Thread#9 running, belonging to process ID 1124",
  "---> Thread#9 over",
  "---> Thread#8 running, belonging to process ID 12700",
  "---> Thread#8 over",
  "---> Thread#7 running, belonging to process ID 2184",
  "---> Thread#7 over",
  "---> Thread#6 running, belonging to process ID 19572",
  "---> Thread#6 over",
  "---> Thread#5 running, belonging to process ID 8672",
  "---> Thread#5 over",
  "---> Thread#4 running, belonging to process ID 15012",
  "---> Thread#4 over",
  "---> Thread#3 running, belonging to process ID 12432",
  "---> Thread#3 over",
  "---> Thread#2 running, belonging to process ID 20748",
  "---> Thread#2 over",
  "---> Thread#1 running, belonging to process ID 14800",
  "---> Thread#1 over",
  "End",
  "--- 9.091314315795898 seconds ---"
]
```

5) Thread synchronization with RLock

سناریو اول:

```python
class MyThread_4(threading.Thread):
    new *
    def __init__(self, thread_id, shared_data, lock):
        super().__init__()
        self.thread_id = thread_id
        self.shared_data = shared_data
        self.lock = lock

    new *
    def run(self):
        while True:
            with self.lock:
                if self.shared_data["items_to_add"] > 0:
                    self.shared_data["output_messages"].append(
                        f"ADDED one item --> {self.shared_data['items_to_add'] - 1} item to ADD"
                    )
                    self.shared_data["items_to_add"] -= 1
                    time.sleep(0.5)
                if self.shared_data["items_to_remove"] > 0:
                    self.shared_data["output_messages"].append(
                        f"REMOVED one item --> {self.shared_data['items_to_remove'] - 1} item to REMOVE"
                    )
                    self.shared_data["items_to_remove"] -= 1
                    time.sleep(0.5)
                else:
                    break
```

حلقه `while True:` ایجاد می‌کند که به طور نامحدود ادامه خواهد یافت مگر اینکه با یک `break` خاتمه یابد. بخش کد داخل `with self.lock:` تنها زمانی اجرا می‌شود که قفل (lock) گرفته شده باشد. اگر `items_to_add` بیشتر از صفر باشد:

یک پیام به `output_messages` اضافه می‌کند که نشان می‌دهد یک آیتم اضافه شده و تعداد آیتم‌های باقی‌مانده برای اضافه کردن را نمایش می‌دهد.

مقدار `items_to_add` را یک واحد کاهش می‌دهد و نخ به مدت 0.5 ثانیه متوقف می‌شود.

اگر `items_to_remove` بیشتر از صفر باشد:

یک پیام به `output_messages` اضافه می‌کند که نشان می‌دهد یک آیتم حذف شده و تعداد آیتم‌های باقی‌مانده برای حذف را نمایش می‌دهد.

مقدار `items_to_remove`را یک واحد کاهش می‌دهد و نخ به مدت 0.5 ثانیه متوقف می‌شود.

اگر هیچ آیتمی برای اضافه کردن یا حذف کردن باقی نمانده باشد، حلقه `while`با `break`خاتمه می‌یابد.

```python
@app.get("/query4/")
async def process_items(items_to_add: int = Query( default: ..., title="Items to Add"),
                        items_to_remove: int = Query( default: ..., title="Items to Remove")):
    lock = threading.RLock()
    shared_data = {
        "items_to_add": items_to_add,
        "items_to_remove": items_to_remove,
        "output_messages": []
    }

    shared_data["output_messages"].append(f"N° {items_to_add} items to ADD")
    shared_data["output_messages"].append(f"N° {items_to_remove} items to REMOVE")

    threads = []
    for i in range(items_to_add):
        thread = MyThread_4(i + 1, shared_data, lock)
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()

    return shared_data["output_messages"]
```

دو پارامتر `items_to_add` و `items_to_remove` از طریق کوئری پارامترهای درخواست دریافت می‌شوند.

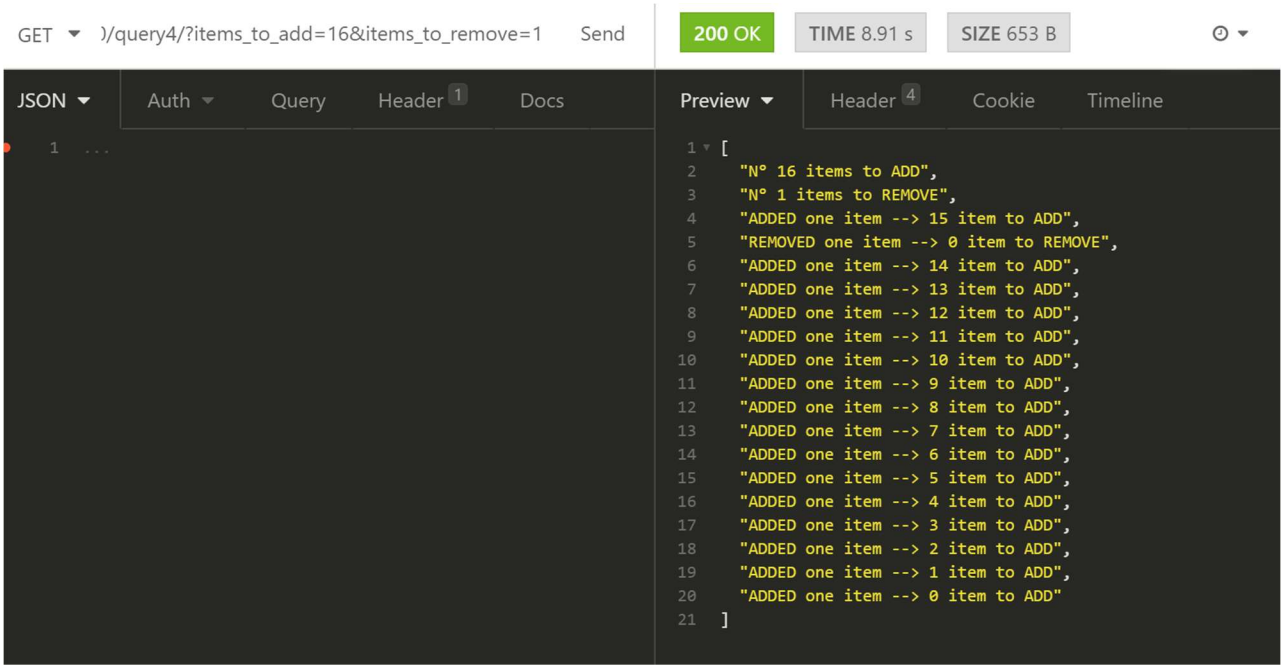
یک قفل بازگشتی (`threading.RLock`) ایجاد می‌کند که می‌تواند توسط همان نخ چند بار گرفته شود.

یک دیکشنری `shared_data` ایجاد می‌کند که شامل تعداد آیتم‌هایی که باید اضافه شوند، (`items_to_add`)، تعداد آیتم‌هایی که باید حذف شوند (`items_to_remove`) و لیستی برای ذخیره پیام‌های خروجی (`output_messages`)

دو پیام ابتدایی به `output_messages` اضافه می‌کند که نشان می‌دهند چند آیتم باید اضافه و حذف شوند.

در هر تکرار یک نخ جدید از نوع `MyThread_4` با شناسه نخ (`thread_id`) برابر با `i + 1` ایجاد می‌کند. نخ را با استفاده از `thread.start()` شروع می‌کند و به لیست `threads` اضافه می‌کند.

http://localhost:8000/query4/?items_to_add=16&items_to_remove=1



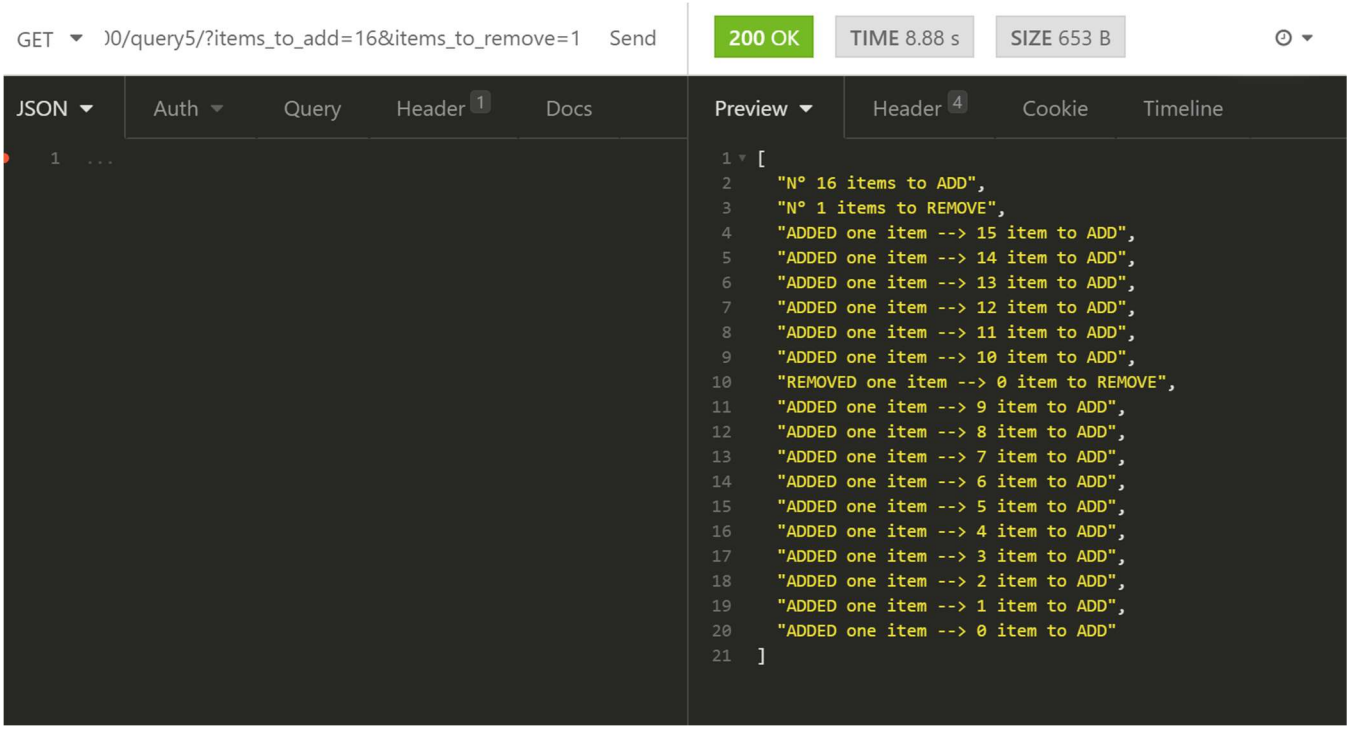
سناریو دوم:

```python
class MyThread_5(threading.Thread):
    new *
    def __init__(self, thread_id, shared_data, lock):
        super().__init__()
        self.thread_id = thread_id
        self.shared_data = shared_data
        self.lock = lock

```

```python
    def run(self):
        add_counter = 0  # ADD شمارنده برای تعداد عملیات

        while True:
            with self.lock:
                if self.shared_data["items_to_add"] > 0:
                    self.shared_data["output_messages"].append(
                        f"ADDED one item --> {self.shared_data['items_to_add'] - 1} item to ADD"
                    )
                    self.shared_data["items_to_add"] -= 1
                    add_counter += 1
                    time.sleep(0.5)

                    # ADD بررسی برای انجام عملیات REMOVE هر 6 عملیات
                    if add_counter == 6 and self.shared_data["items_to_remove"] > 0:
                        self.shared_data["output_messages"].append(
                            f"REMOVED one item --> {self.shared_data['items_to_remove'] - 1} item to REMOVE"
                        )
                        self.shared_data["items_to_remove"] -= 1
                        add_counter = 0  # تنظیم مجدد شمارنده برای ADD به صفر
                        time.sleep(0.5)

                elif self.shared_data["items_to_remove"] > 0:
                    self.shared_data["output_messages"].append(
                        f"REMOVED one item --> {self.shared_data['items_to_remove'] - 1} item to REMOVE"
                    )
                    self.shared_data["items_to_remove"] -= 1
                    time.sleep(0.5)
                else:
                    break
```

```python
@app.get("/query5/")
async def process_items(items_to_add: int = Query( default: ..., title="Items to Add"),
                        items_to_remove: int = Query( default: ..., title="Items to Remove")):
    lock = threading.RLock()
    shared_data = {
        "items_to_add": items_to_add,
        "items_to_remove": items_to_remove,
        "output_messages": []
    }

    shared_data["output_messages"].append(f"N° {items_to_add} items to ADD")
    shared_data["output_messages"].append(f"N° {items_to_remove} items to REMOVE")

    threads = []
    for i in range(items_to_add):
        thread = MyThread_5(i + 1, shared_data, lock)
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()

    return shared_data["output_messages"]

```

یک شمارنده `add_counter` برای شمارش تعداد عملیات اضافه کردن (ADD) ایجاد می‌کند که باعث میشود هر 6 عملیات اضافه کردن، یک عملیات حذف کردن انجام شود.



سناریو سوم:

```python
class MyThread_6(threading.Thread):
    new *
    def __init__(self, thread_id, shared_data, lock):
        super().__init__()
        self.thread_id = thread_id
        self.shared_data = shared_data
        self.lock = lock

    new *
    def run(self):
        while True:
            with self.lock:
                if self.shared_data["items_to_add"] > 0:
                    self.shared_data["output_messages"].append(
                        f"ADDED one item --> {self.shared_data['items_to_add'] - 1} item to ADD"
                    )
                    self.shared_data["items_to_add"] -= 1
                    time.sleep(0.5)
                elif self.shared_data["items_to_remove"] > 0:
                    self.shared_data["output_messages"].append(
                        f"REMOVED one item --> {self.shared_data['items_to_remove'] - 1} item to REMOVE"
                    )
                    self.shared_data["items_to_remove"] -= 1
                    time.sleep(0.5)
                else:
                    break
```

```python
@app.get("/query6/")
async def process_items(items_to_add: int = Query( default: ..., title="Items to Add"),
                        items_to_remove: int = Query( default: ..., title="Items to Remove")):
    lock = threading.RLock()
    shared_data = {
        "items_to_add": items_to_add,
        "items_to_remove": items_to_remove,
        "output_messages": []
    }

    shared_data["output_messages"].append(f"N° {items_to_add} items to ADD")
    shared_data["output_messages"].append(f"N° {items_to_remove} items to REMOVE")

    threads = []
    total_operations = items_to_add + items_to_remove

    for i in range(total_operations):
        thread = MyThread_6(i + 1, shared_data, lock)
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()

    return shared_data["output_messages"]
```

این سناریو همان سناریو اول است با این تفاوت که ابتدا همه ی نخ ها اضافه میشوند و در اخر یک نخ حذف میشود.

که این با تغییر if به elif در تابع run پدید امده است.

GET)/query6/?items_to_add=16&items_to_remove=1 Send

200 OK | TIME 8.84 s | SIZE 653 B

JSON | Auth | Query | Header 1 | Docs

```
1 ...
```

Preview | Header 4 | Cookie | Timeline


```
1  [
2    "N° 16 items to ADD",
3    "N° 1 items to REMOVE",
4    "ADDED one item --> 15 item to ADD",
5    "ADDED one item --> 14 item to ADD",
6    "ADDED one item --> 13 item to ADD",
7    "ADDED one item --> 12 item to ADD",
8    "ADDED one item --> 11 item to ADD",
9    "ADDED one item --> 10 item to ADD",
10   "ADDED one item --> 9 item to ADD",
11   "ADDED one item --> 8 item to ADD",
12   "ADDED one item --> 7 item to ADD",
13   "ADDED one item --> 6 item to ADD",
14   "ADDED one item --> 5 item to ADD",
15   "ADDED one item --> 4 item to ADD",
16   "ADDED one item --> 3 item to ADD",
17   "ADDED one item --> 2 item to ADD",
18   "ADDED one item --> 1 item to ADD",
19   "ADDED one item --> 0 item to ADD",
20   "REMOVED one item --> 0 item to REMOVE"
21 ]
```

6) Thread synchronization with semaphores

سناریو اول:

```
518 semaphore = threading.Semaphore(0)
519 max_notifications = 10
520 output_list = []
521 output_list_lock = threading.Lock()
522
    1 usage  new *
523 class Producer(threading.Thread):
        new *
524     def run(self):
525         for _ in range(max_notifications):
526             time.sleep(1)
527             item_number = random.randint(100, 999)
528             with output_list_lock:
529                 output_list.append(f"{time.strftime('%Y-%m-%d %H:%M:%S')} "
530                                    f"{self.name} INFO Producer notify: item number {item_number}")
531             semaphore.release()  # Notify one consumer
```

خط 518 یک semaphore ایجاد می‌کند که ابتدا با مقدار 0 مقداردهی شده است، semaphore یک مکانیزم همگام‌سازی (synchronization) است که برای کنترل دسترسی به منابع مشترک بین نخ‌ها استفاده می‌شود. خط 519 تعداد اعلان‌ها (notifications) را که هر تولیدکننده تولید می‌کند، مشخص می‌نماید. خطوط 520 و 521 یک لیست خروجی برای ذخیره‌سازی پیام‌های لاگ (log) و یک قفل (lock) برای همگام‌سازی دسترسی به این لیست ایجاد می‌کنند.

کلاس Producer یک نخ تولیدکننده را تعریف می‌کند. متد `run` این کلاس شامل یک حلقه است که:

یک ثانیه منتظر می‌ماند، یک عدد تصادفی بین 100 و 999 تولید می‌کند، با استفاده از قفل، پیام لاگ تولید می‌کند و به لیست خروجی اضافه می‌کند و یک واحد به semaphore اضافه می‌کند تا یک مصرف‌کننده را آگاه سازد.

```
533 class Consumer(threading.Thread):
        new *
534     def run(self):
535         for _ in range(max_notifications):
536             with output_list_lock:
537                 output_list.append(f"{time.strftime('%Y-%m-%d %H:%M:%S')} "
538                                    f"{self.name} INFO Consumer is waiting")
539             semaphore.acquire()  # Wait for a notification
540             with output_list_lock:
541                 # Process the item
542                 last_notification = next(
543                     (msg for msg in reversed(output_list) if "Producer notify" in msg), None
544                 )
545                 if last_notification:
546                     item_number = last_notification.split()[-1]
547                     output_list.append(f"{time.strftime('%Y-%m-%d %H:%M:%S')} "
548                                        f"{self.name} INFO Consumer notify: item number {item_number}")
```

کلاس Consumer یک نخ مصرف‌کننده را تعریف می‌کند. متد `run` این کلاس شامل یک حلقه است که:

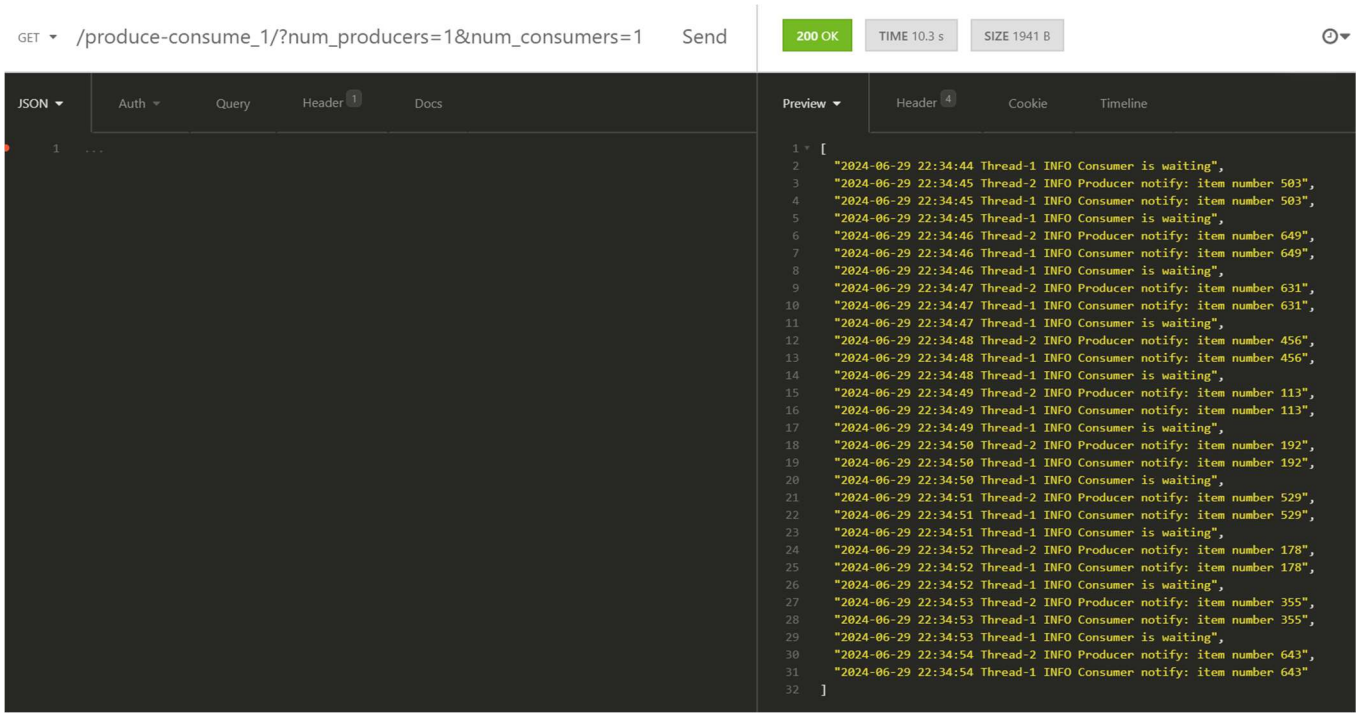
با استفاده از قفل، پیام "در انتظار" را به لیست خروجی اضافه می‌کند، منتظر یک اعلان (notification) از طریق semaphore می‌شود، پس از دریافت اعلان، با استفاده از قفل، آخرین پیام تولیدکننده را پیدا کرده و شماره آیتم (item number) را استخراج می‌کند و پیام "اعلان مصرف‌کننده" را به لیست خروجی اضافه می‌کند.

```python
@app.get("/produce-consume_1/", response_model=List[str])
async def produce_consume(
    num_producers: Optional[int] = Query(default=1, ge=1, le=10),
    num_consumers: Optional[int] = Query(default=1, ge=1, le=10)
):
    global output_list
    output_list = []

    # Start consumer threads
    consumers = []
    for i in range(num_consumers):
        consumer = Consumer(name=f"Thread-{i*2+1}")
        consumers.append(consumer)
        consumer.start()

    # Start producer threads
    producers = []
    for i in range(num_producers):
        producer = Producer(name=f"Thread-{i*2+2}")
        producers.append(producer)
        producer.start()

    for producer in producers:
        producer.join()
    for consumer in consumers:
        consumer.join()

    return output_list
```

پارامترهای ورودی برای تعداد تولیدکننده‌ها و مصرف‌کننده‌ها را می‌گیرد و لیست خروجی را خالی می‌کند.

نخ‌های مصرف‌کننده را ایجاد و شروع می‌کند، نخ‌های تولیدکننده را ایجاد و شروع می‌کند. منتظر پایان کار تمامی تولیدکننده‌ها و مصرف‌کننده‌ها می‌ماند و لیست خروجی را بازمی‌گرداند.

http://localhost:7000/produce-consume_2/?num_producers=1&num_consumers=1



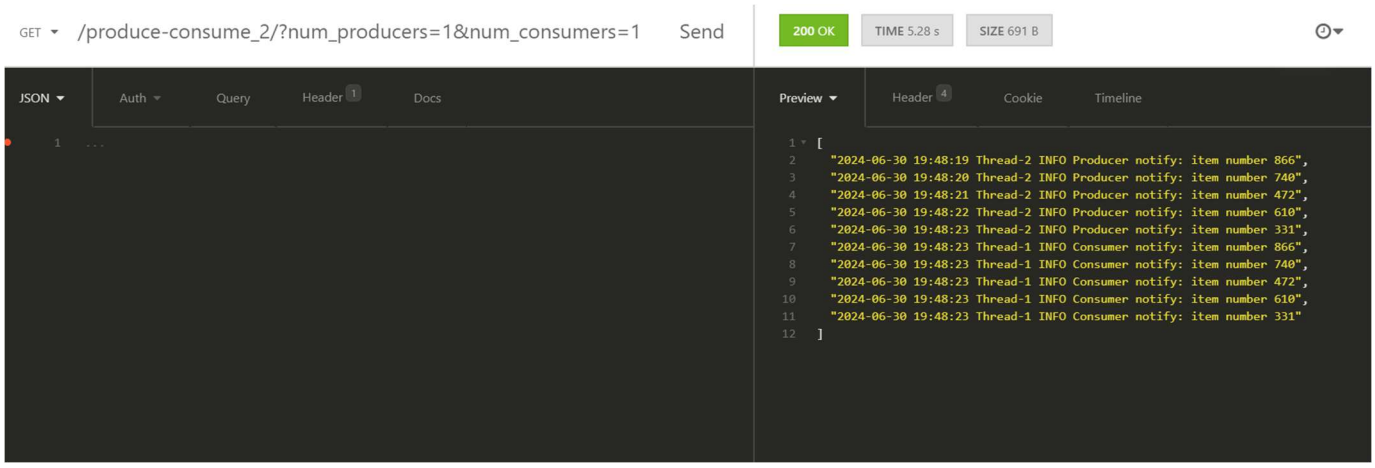
سناریو دوم:

```python
semaphore_items = threading.Semaphore(0)  # کنترل دسترسی مصرف‌کنندگان به آیتم‌ها
semaphore_producers_done = threading.Semaphore(0)  # اطمینان از اتمام تولیدکنندگان
max_notifications = 5
output_list_2 = []
output_list_lock_2 = threading.Lock()
item_queue = Queue()

1 usage  new *
class Producer_2(threading.Thread):
    new *
    def run(self):
        for _ in range(max_notifications):
            time.sleep(1)
            item_number = random.randint(100, 999)
            with output_list_lock_2:
                output_list_2.append(f"{time.strftime('%Y-%m-%d %H:%M:%S')} "
                                     f"{self.name} INFO Producer notify: item number {item_number}")
                item_queue.put(item_number)
            semaphore_items.release()  # Notify one consumer
        semaphore_producers_done.release()  # Signal producer is done

```

```python
class Consumer_2(threading.Thread):
    new *
    def run(self):
        # Wait for all producers to finish
        semaphore_producers_done.acquire()
        for _ in range(max_notifications):
            semaphore_items.acquire()  # Wait for a notification
            item_number = item_queue.get()
            with output_list_lock_2:
                output_list_2.append(f"{time.strftime('%Y-%m-%d %H:%M:%S')} "
                                     f"{self.name} INFO Consumer notify: item number {item_number}")

```

```python
@app.get("/produce-consume_2/", response_model=List[str])
async def produce_consume(
    num_producers: Optional[int] = Query(default=1, ge=1, le=10),
    num_consumers: Optional[int] = Query(default=1, ge=1, le=10)
):
    global output_list_2
    output_list_2 = []
    item_queue.queue.clear()  # Clear the queue for new run

    producers_2 = []
    for i in range(num_producers):
        producer = Producer_2(name=f"Thread-{i*2+2}")
        producers_2.append(producer)
        producer.start()
    for producer in producers_2:
        producer.join()

    # Signal consumers that producers are finished
    semaphore_producers_done.release()

    consumers_2 = []
    for i in range(num_consumers):
        consumer = Consumer_2(name=f"Thread-{i*2+1}")
        consumers_2.append(consumer)
        consumer.start()
    for consumer in consumers_2:
        consumer.join()

    return output_list_2
```

در این سناریو ابتدا اول همه تولید کننده ها کار خود را تمام میکنند و بعد مصرف کننده ها شروع به مصرف میکنند با تغییر جای join ها در تولید کننده ها به این سناریو دست یافتیم.



7) Thread synchronization with a barrier

سناریو اول:

```
650 barrier = Barrier(3)  # Create a barrier for 3 threads
651
    1 usage  new *
652 def race_participant(name):
653     output_messages.append(f"{name} reached the barrier at: {datetime.datetime.now()}")
654     barrier.wait()
655     output_messages.append(f"{name} passed the barrier at: {datetime.datetime.now()}")
656
    new *
657 @app.get("/barrier_1")
658 async def start_race():
659     global output_messages
660     output_messages = []
661     output_messages.append("START RACE!!!!")
662
663     # Define participants
664     participants = ["Dewey", "Huey", "Louie"]
665     threads = []
666
667     for participant in participants:
668         thread = threading.Thread(target=race_participant, args=(participant,))
669         threads.append(thread)
670         thread.start()
671     for thread in threads:
672         thread.join()
673
674     output_messages.append("Race over!")
675
676     return output_messages
```

خط 650 یک Barrier برای 3 نخ ایجاد می‌کند. Barrier مکانیزمی است که به تعداد معینی از نخ‌ها اجازه می‌دهد تا در نقطه‌ای منتظر بمانند تا همه آنها به آن نقطه برسند و سپس همه به طور همزمان ادامه دهند.

تابع race_participant وظیفه شرکت‌کننده در مسابقه را تعریف می‌کند. هر شرکت‌کننده:

یک پیام را به لیست `output_messages` اضافه می‌کند که نشان می‌دهد شرکت‌کننده به مانع رسیده است، به همراه زمان فعلی و منتظر می‌ماند تا همه شرکت‌کنندگان به مانع برسند با استفاده از `barrier.wait()`

پس از گذر از مانع، یک پیام دیگر به لیست `output_messages` اضافه می‌کند که نشان می‌دهد شرکت‌کننده از مانع عبور کرده است، به همراه زمان فعلی.

در خط 661 یک پیام "START RACE!!!!" به لیست اضافه می‌شود تا شروع مسابقه را نشان دهد.

خطوط 664 و 665 شرکت‌کنندگان در مسابقه را تعریف می‌کند (Dewey, Huey, Louie) و یک لیست برای نگهداری نخ‌های مربوط به هر شرکت‌کننده ایجاد می‌کند.

در حلقه برای هر شرکت‌کننده یک نخ ایجاد می‌کند که تابع `race_participant` را با نام شرکت‌کننده به عنوان آرگومان فراخوانی می‌کند.

نخ را به لیست `threads` اضافه می‌کند و نخ را شروع می‌کند.

حلقه خط 671 منتظر می‌ماند تا همه نخ‌ها به پایان برسند (همه شرکت‌کنندگان مسابقه را تمام کنند).

خطوط 674 و 676 یک پیام "Race over!" به لیست `output_messages` اضافه می‌کنند تا پایان مسابقه را نشان دهند و لیست `output_messages` را به عنوان خروجی endpoint بازمی‌گردانند.

http://localhost:7000/barrier

GET http://localhost:8000/barrier Send

200 OK TIME 250 ms SIZE 380 B

JSON Auth Query Header 1 Docs

```
1 ...
```

Preview Header 4 Cookie Timeline


```
1 [
2   "START RACE!!!!",
3   "Dewey reached the barrier at: 2024-06-30 21:33:30.087760",
4   "Huey reached the barrier at: 2024-06-30 21:33:30.087760",
5   "Louie reached the barrier at: 2024-06-30 21:33:30.088693",
6   "Louie passed the barrier at: 2024-06-30 21:33:30.088693",
7   "Huey passed the barrier at: 2024-06-30 21:33:30.088693",
8   "Dewey passed the barrier at: 2024-06-30 21:33:30.088693",
9   "Race over!"
10 ]
```

Process-Based Parallelism

1) Spawning a process

سناریو اول:

```
684 def myFunc(n, idx, shared_list):
685     output = []
686     output.append(f"calling myFunc from process n°: {idx}")
687     for i in range(n):
688         output.append(f"output from myFunc is :{i}")
689     shared_list[idx] = output
    new *
690 @app.get("/calculate")
691 async def calculate(
692     n0: int = Query(...),
693     n1: int = Query(...),
694     n2: int = Query(...),
695     n3: int = Query(...),
696     n4: int = Query(...),
697     n5: int = Query(...)
698 ):
699     inputs = [n0, n1, n2, n3, n4, n5]
700     manager = multiprocessing.Manager()
701     shared_list = manager.list([None] * len(inputs))
702     processes = []
703
704     for idx, n in enumerate(inputs):
705         p = multiprocessing.Process(target=myFunc, args=(n, idx, shared_list))
706         processes.append(p)
707         p.start()
708     for p in processes:
709         p.join()
710
711     results = list(shared_list)
712     return results
```

`myFunc` یک تابع است که تعدادی ورودی از جمله `n` که یک عدد صحیح است و `idx` که شاخص فرآیند و `shared_list` که یک لیست اشتراکی برای ذخیره نتایج است دریافت می‌کند. این تابع یک لیست `output` ایجاد کرده و پیام‌هایی را به آن اضافه می‌کند، سپس نتایج حاصل از تابع را در `shared_list` بر اساس `idx` ذخیره می‌کند.

در خط 690 یک وب‌سرویس با نام `calculate` تعریف شده است که به ورودی‌های `n0` تا `n5` نیاز دارد که از پارامترهای Query دریافت می‌شوند.

این وب‌سرویس ابتدا ورودی‌ها را در یک لیست `inputs` ذخیره می‌کند، سپس یک نمونه از `multiprocessing.Manager` به نام `manager` ایجاد می‌شود و یک لیست اشتراکی به نام `shared_list` با اندازه مشابه `inputs` ساخته می‌شود.

در ادامه، برای هر `n` در `inputs`، یک فرآیند جدید ایجاد می‌شود که به عنوان هدف تابع `myFunc` را دارد و پارامترهای لازم را ارسال می‌کند.

همه فرآیندها با استفاده از `.start()` شروع می‌شوند و سپس با استفاده از `.join()` منتظر پایان آنها می‌ماند.

در نهایت، `shared_list` به یک لیست عادی تبدیل شده و به عنوان نتیجه برگردانده می‌شود.

http://localhost:7000/calculate?n0=0&n1=1&n2=2&n3=3&n4=4&n5=5

GET 8000/calculate?n0=0&n1=1&n2=2&n3=3&n4=4&n5=5 Send

200 OK TIME 2.84 s SIZE 640 B

JSON Auth Query Header 1 Docs

Preview Header 4 Cookie Timeline


```
[
  "calling myFunc from process n°: 0"
],
[
  "calling myFunc from process n°: 1",
  "output from myFunc is :0"
],
[
  "calling myFunc from process n°: 2",
  "output from myFunc is :0",
  "output from myFunc is :1"
],
[
  "calling myFunc from process n°: 3",
  "output from myFunc is :0",
  "output from myFunc is :1",
  "output from myFunc is :2"
],
[
  "calling myFunc from process n°: 4",
  "output from myFunc is :0",
  "output from myFunc is :1",
  "output from myFunc is :2",
  "output from myFunc is :3"
],
[
  "calling myFunc from process n°: 5",
  "output from myFunc is :0",
  "output from myFunc is :1",
  "output from myFunc is :2",
  "output from myFunc is :3",
  "output from myFunc is :4"
]
]
```


2) Naming a process

سناریو اول:

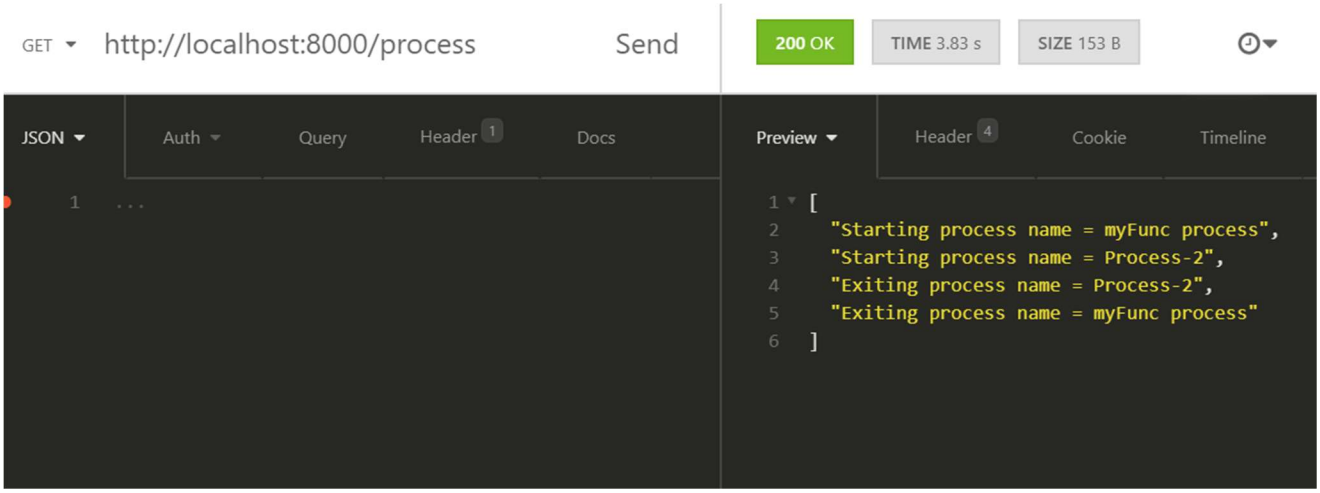
```python
def myFunc_process(queue,sleepTime):
    proc = current_process()
    queue.put(f"Starting process name = {proc.name}")
    time.sleep(sleepTime)  # Simulate some process work
    queue.put(f"Exiting process name = {proc.name}")


new *
@app.get("/process")
def run_process(name1: str = Query("myFunc process"), name2: str = Query("Process-2")):
    output_queue = Queue()

    p1 = Process(name=name1, target=myFunc_process, args=(output_queue,2))
    p2 = Process(name=name2, target=myFunc_process, args=(output_queue,1))

    p1.start()
    p2.start()

    p1.join()
    p2.join()

    output_messages = []
    while not output_queue.empty():
        output_messages.append(output_queue.get())

    return output_messages
```

myFunc_process یک تابع است که دو ورودی می‌گیرد: queue یک صف اشتراکی برای ذخیره پیام‌ها و sleepTime زمان تأخیر برای شبیه‌سازی فعالیت فرآیند. ابتدا نمونه فرآیند جاری را دریافت می‌کند و پیام‌های "شروع فرآیند" و "خروج فرآیند" را به صف اشتراکی (queue) اضافه می‌کند.

این وب‌سرویس دو ورودی name1 و name2 را از پارامترهای Query دریافت می‌کند که به ترتیب نام فرآیندهای ایجاد شده هستند.

ابتدا یک Queue برای اشتراک داده‌ها بین فرآیندها ایجاد می‌شود سپس دو فرآیند p1 و p2 ایجاد می‌شوند، هرکدام با یک نام و هدف تابع myFunc_process و پارامترهای مختلف (sleepTime)

http://localhost:7000/process



سناریو دوم:

```python
@app.get("/process_2")
def run_process(name1: str = Query("myFunc process"), name2: str = Query("Process-2")):
    output_queue = Queue()

    p1 = Process(name=name1, target=myFunc_process, args=(output_queue,2))
    p2 = Process(name=name2, target=myFunc_process, args=(output_queue,1))

    p1.start()
    p1.join()
    p2.start()
    p2.join()

    output_messages = []
    while not output_queue.empty():
        output_messages.append(output_queue.get())

    return output_messages
```

GET http://localhost:8000/process_2 Send

200 OK TIME 6.28 s SIZE 153 B

JSON Auth Query Header 1 Docs

```
1 ...
```

Preview Header 4 Cookie Timeline

```
1 [
2   "Starting process name = myFunc process",
3   "Exiting process name = myFunc process",
4   "Starting process name = Process-2",
5   "Exiting process name = Process-2"
6 ]
```

سناریو سوم:

```python
@app.get("/process_3")
def run_process(name1: str = Query("myFunc process"), name2: str = Query("Process-2")):
    output_queue = Queue()

    p1 = Process(name=name1, target=myFunc_process, args=(output_queue,1))
    p2 = Process(name=name2, target=myFunc_process, args=(output_queue,1))

    p1.start()
    p2.start()
    p1.join()
    p2.join()

    output_messages = []
    while not output_queue.empty():
        output_messages.append(output_queue.get())

    return output_messages
```

GET http://localhost:8000/process_3 Send

200 OK TIME 2.94 s SIZE 153 B

JSON Auth Query Header 1 Docs

```
1 ...
```

Preview Header 4 Cookie Timeline

```
1 [
2   "Starting process name = Process-2",
3   "Starting process name = myFunc process",
4   "Exiting process name = Process-2",
5   "Exiting process name = myFunc process"
6 ]
```

3) Running processes in the background

سناریو اول:

```python
def background_process(start, end, queue):
    queue.put("Starting background_process")
    for i in range(start, end):
        queue.put(f"---> {i}")
        time.sleep(0.5)  # ایجاد تاخیر برای مشاهده ترتیب اجرا
    queue.put("Exiting background_process")

2 usages  new *
def no_background_process(start, end, queue):
    queue.put("Starting NO_background_process")
    for i in range(start, end):
        queue.put(f"---> {i}")
        time.sleep(0.5)  # ایجاد تاخیر برای مشاهده ترتیب اجرا
    queue.put("Exiting NO_background_process")
```

تابع `background_process` یک فرآیند را باز می‌کند که شروع و خروج آن را به همراه تمامی پیام‌های `---> i` بین شماره `start` تا `end-1` به یک صف (`queue`) اضافه می‌کند. هر پیام با یک تاخیر 0.5 ثانیه قرار داده می‌شود تا بتوان ترتیب اجرا را مشاهده کرد.

تابع `no_background_process` همانند `background_process` عمل می‌کند، با این تفاوت که فرآیند آن به صورت همزمان با صف بندی نشده (بدون پس‌زمینه) اجرا می‌شود.

```python
new *
@app.get("/run_processes_1/")
async def run_processes():
    queue = Queue()

    p1 = Process(target=no_background_process, args=(5, 10, queue))
    p2 = Process(target=no_background_process, args=(5, 10, queue))
    p3 = Process(target=background_process, args=(0, 5, queue))

    p1.start()
    p1.join()

    p2.start()
    p3.start()

    p2.join()
    p3.join()

    output = []
    while not queue.empty():
        output.append(queue.get())
```

در اینجا یک `Queue`ایجاد شده و سه فرآیند ایجاد می‌شود:

دو فرآیند `no_background_process` بدون پس‌زمینه (همزمان) و یک فرآیند `background_process` با پس‌زمینه.

http://localhost:7000/run_processes

GET http://localhost:8000/run_processes Send

200 OK TIME 8.03 s SIZE 325 B

JSON Auth Query Header 1 Docs

Preview Header 4 Cookie Timeline


```
[
  "Starting NO_background_process",
  "---> 5",
  "---> 6",
  "---> 7",
  "---> 8",
  "---> 9",
  "Exiting NO_background_process",
  "Starting NO_background_process",
  "---> 5",
  "Starting background_process",
  "---> 0",
  "---> 1",
  "---> 6",
  "---> 2",
  "---> 7",
  "---> 3",
  "---> 8",
  "---> 9",
  "---> 4",
  "Exiting background_process",
  "Exiting NO_background_process"
]
```

سناریو دوم:

```python
new *
@app.get("/run_processes_2/")
async def run_processes():
    queue = Queue()

    p1 = Process(target=no_background_process, args=(5, 10, queue))
    p2 = Process(target=no_background_process, args=(5, 10, queue))
    p3 = Process(target=background_process, args=(0, 5, queue))

    p1.start()
    p1.join()

    p2.start()
    p2.join()
    p3.start()
    p3.join()

    output = []
    while not queue.empty():
        output.append(queue.get())

    return output
```

GET http://localhost:8000/run_processes Send

200 OK TIME 11.7 s SIZE 325 B

JSON Auth Query Header 1 Docs

```
...
```

Preview Header 4 Cookie Timeline

```
[
  "Starting NO_background_process",
  "---> 5",
  "---> 6",
  "---> 7",
  "---> 8",
  "---> 9",
  "Exiting NO_background_process",
  "Starting NO_background_process",
  "---> 5",
  "---> 6",
  "---> 7",
  "---> 8",
  "---> 9",
  "Exiting NO_background_process",
  "Starting background_process",
  "---> 0",
  "---> 1",
  "---> 2",
  "---> 3",
  "---> 4",
  "Exiting background_process"
]
```

4) Killing a process

سناریو اول:

```python
847 @app.get("/Killing_process_1/")
848 async def manage_process():
849     process = Process()
850     process_before = (str(process), process.is_alive())
851
852     process.start()
853     process_running = (str(process), process.is_alive())
854
855     process.terminate()
856     process_terminated = (str(process), process.is_alive())
857
858     process.join()
859     process_joined = (str(process), process.is_alive())
860
861     process_exit_code = process.exitcode
862
863     response = {
864         "Process before execution": f"<Process({process.name}, initial)> {process_before[1]}",
865         "Process running": f"<Process({process.name}, started)> {process_running[1]}",
866         "Process terminated": f"<Process({process.name}, started)> {process_terminated[1]}",
867         "Process joined": f"<Process({process.name}, stopped[SIGTERM])> {process_joined[1]}",
868         "Process exit code": process_exit_code
869     }
870     return response
871
```

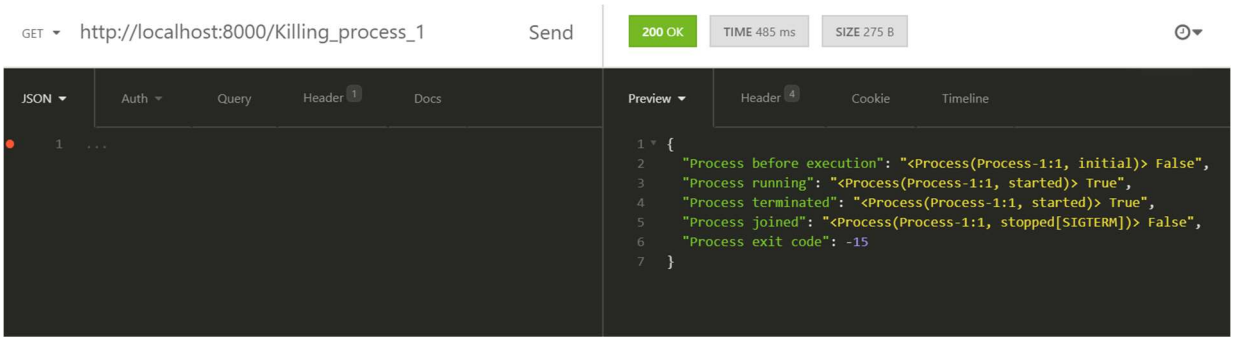
ابتدا در خط 850 وضعیت فرایند و اطلاعات آن ذخیره میشود و بعد در خط 852 فرایند شروع میشود و دوباره وضعیت و اطلاعاتش ذخیره میشوند.

در خط 855 فرایند متوقف میشود و در خط بعد وضعیت و اطلاعاتش ذخیره میشوند.

در خط 858 منتظر میماند تا فرایند تمام شود و بعد وضعیت و اطلاعاتش ذخیره میشوند.

در خط 861 کد خروج فرایند ذخیره میشود.

http://localhost:7000/killing_processes_1



چون سناریو با توقف پایان یافته کد خروج منفی است.

سناریو دوم:

```python
@app.get("/Killing_process_2/")
async def manage_process():
    process = Process()
    process_before = (str(process), process.is_alive())

    process.start()
    process_running = (str(process), process.is_alive())

    # process.terminate()
    # process_terminated = (str(process), process.is_alive())

    process.join()
    process_joined = (str(process), process.is_alive())

    process_exit_code = process.exitcode

    response = {
        "Process before execution": f"<Process({process.name}, initial)> {process_before[1]}",
        "Process running": f"<Process({process.name}, started)> {process_running[1]}",
        # "Process terminated": f"<Process({process.name}, started)> {process_terminated[1]}",
        "Process joined": f"<Process({process.name}, stopped[SIGTERM])> {process_joined[1]}",
        "Process exit code": process_exit_code
    }
    return response
```

GET http://localhost:8000/Killing_process_2 Send

200 OK TIME 625 ms SIZE 213 B

JSON Auth Query Header 1 Docs

```
1 ...
```

Preview Header 4 Cookie Timeline

```
1 {
2   "Process before execution": "<Process(Process-1:1, initial)> False",
3   "Process running": "<Process(Process-1:1, started)> True",
4   "Process joined": "<Process(Process-1:1, stopped[SIGTERM])> False",
5   "Process exit code": 0
6 }
```

چون سناریو خودش تمام شده با کد خروج صفر پایان یافته است.

5) Defining processes in a subclass

سناریو اول:

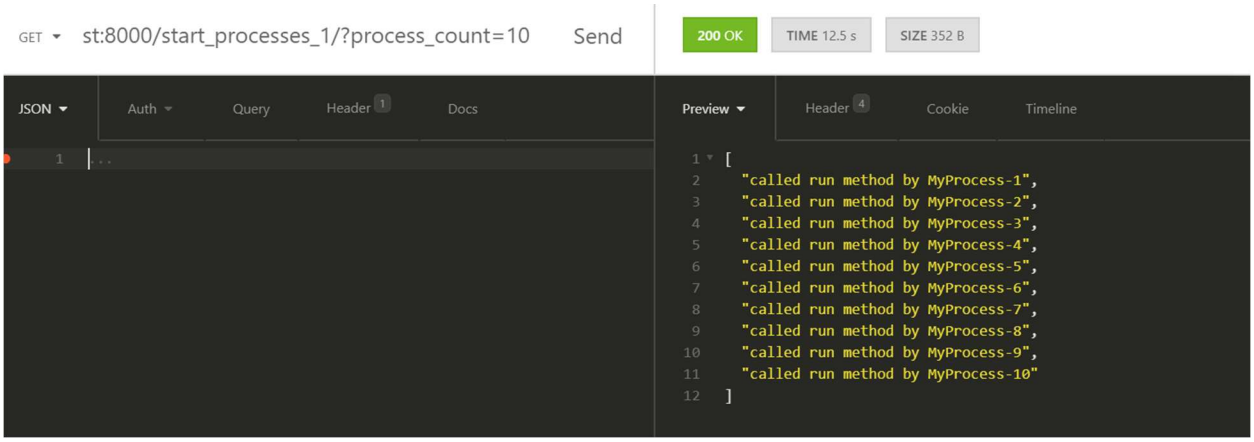
```python
904 class MyProcess(multiprocessing.Process):
        new *
905     def __init__(self, process_id, queue):
906         super().__init__()
907         self.process_id = process_id
908         self.queue = queue
909
        new *
910     def run(self):
911         self.queue.put(f"called run method by MyProcess-{self.process_id}")
```

MyProcess این یک زیرکلاس از multiprocessing.Process است که برای ایجاد فرآیندها استفاده می‌شود. در این کلاس، متد __init__ برای مقداردهی اولیه اجرا می‌شود که process_id و queue را به عنوان ورودی می‌گیرد. process_id شناسه یکتای هر فرآیند است و queue یک صف است که برای ارسال پیام به فرآیند استفاده می‌شود. در متد run، که از Process ارث‌بری شده است، پیامی که نشان‌دهنده شروع اجرای فرآیند است، به queue ارسال می‌شود.

```python
913 @app.get("/start_processes_1/")
914 async def start_processes(process_count: int = Query(default=10, ge=1, le=10)):
915     queue = multiprocessing.Queue()
916     processes = []
917
918     for i in range(1, process_count + 1):
919         process = MyProcess(i, queue)
920         processes.append(process)
921         process.start()
922         process.join()
923
924     results = []
925     while not queue.empty():
926         results.append(queue.get())
927
928     return results
929
```

در خط 915 یک صف Queue برای ارتباط بین فرآیندها ایجاد می‌شودو در خط 916 یک لیست برای ذخیره فرآیندها ایجاد می‌شود.

http://localhost:7000/start_processes_1/?process_count=10



سناریو دوم:

```python
@app.get("/start_processes_2/")
async def start_processes(process_count: int = Query(default=10, ge=1, le=10)):
    queue = multiprocessing.Queue()
    processes = []

    for i in range(1, process_count + 1):
        process = MyProcess(i, queue)
        processes.append(process)
        process.start()

    for process in processes:
        process.join()

    results = []
    while not queue.empty():
        results.append(queue.get())

    return results
```

GET ost:8000/start_processes_2/?process_count=10 Send

200 OK TIME 3 s SIZE 352 B

JSON Auth Query Header 1 Docs

```
1 ...
```

Preview Header 4 Cookie Timeline

```
1  [
2    "called run method by MyProcess-6",
3    "called run method by MyProcess-9",
4    "called run method by MyProcess-2",
5    "called run method by MyProcess-4",
6    "called run method by MyProcess-1",
7    "called run method by MyProcess-5",
8    "called run method by MyProcess-7",
9    "called run method by MyProcess-3",
10   "called run method by MyProcess-10",
11   "called run method by MyProcess-8"
12 ]
```

سناریو سوم:

```python
@app.get("/start_processes_3/")
async def start_processes(process_count: int = Query(default=10, ge=1, le=10)):
    queue = multiprocessing.Queue()
    processes = []

    for i in range(process_count, 0, -1):
        process = MyProcess(i, queue)
        processes.append(process)
        process.start()
        process.join()

    results = []
    while not queue.empty():
        results.append(queue.get())

    return results
```

GET localhost:8000/start_processes_3/?process_cou Send

200 OK TIME 12.6 s SIZE 352 B

JSON Auth Query Header 1 Docs

```
1 ...
```

Preview Header 4 Cookie Timeline

```
1  [
2    "called run method by MyProcess-10",
3    "called run method by MyProcess-9",
4    "called run method by MyProcess-8",
5    "called run method by MyProcess-7",
6    "called run method by MyProcess-6",
7    "called run method by MyProcess-5",
8    "called run method by MyProcess-4",
9    "called run method by MyProcess-3",
10   "called run method by MyProcess-2",
11   "called run method by MyProcess-1"
12 ]
```

6) Using a queue to exchange data

سناریو اول:

```python
def producer(queue, log_queue, producer_id):
    for _ in range(10):
        item = random.randint(1, 300)
        queue.put((item, producer_id))
        log_queue.put(f"Process Producer : item {item} appended to queue producer-{producer_id}")
        log_queue.put(f"The size of queue is {queue.qsize()}")
        time.sleep(0.5)

1 usage  new *
def consumer(queue, log_queue, consumer_id):
    while True:
        if not queue.empty():
            item, producer_id = queue.get()
            log_queue.put(f"Process Consumer : item {item} popped from queue by consumer-{consumer_id}")
            time.sleep(1.5)
        else:
            time.sleep(0.1)  # Short sleep to avoid busy-waiting
```

تابع producer یک تولیدکننده را نمایش می‌دهد که برای ۱۰ بار یک عدد تصادفی بین ۱ تا ۳۰۰ تولید می‌کند و آن را به صف queue اضافه می‌کند. همچنین پیام‌هایی را در log_queue قرار می‌دهد که وضعیت تولید را نمایش می‌دهند و اندازه فعلی صف queue را نیز نمایش می‌دهد. سپس با استفاده از time.sleep(0.5) تاخیری ایجاد می‌کند.

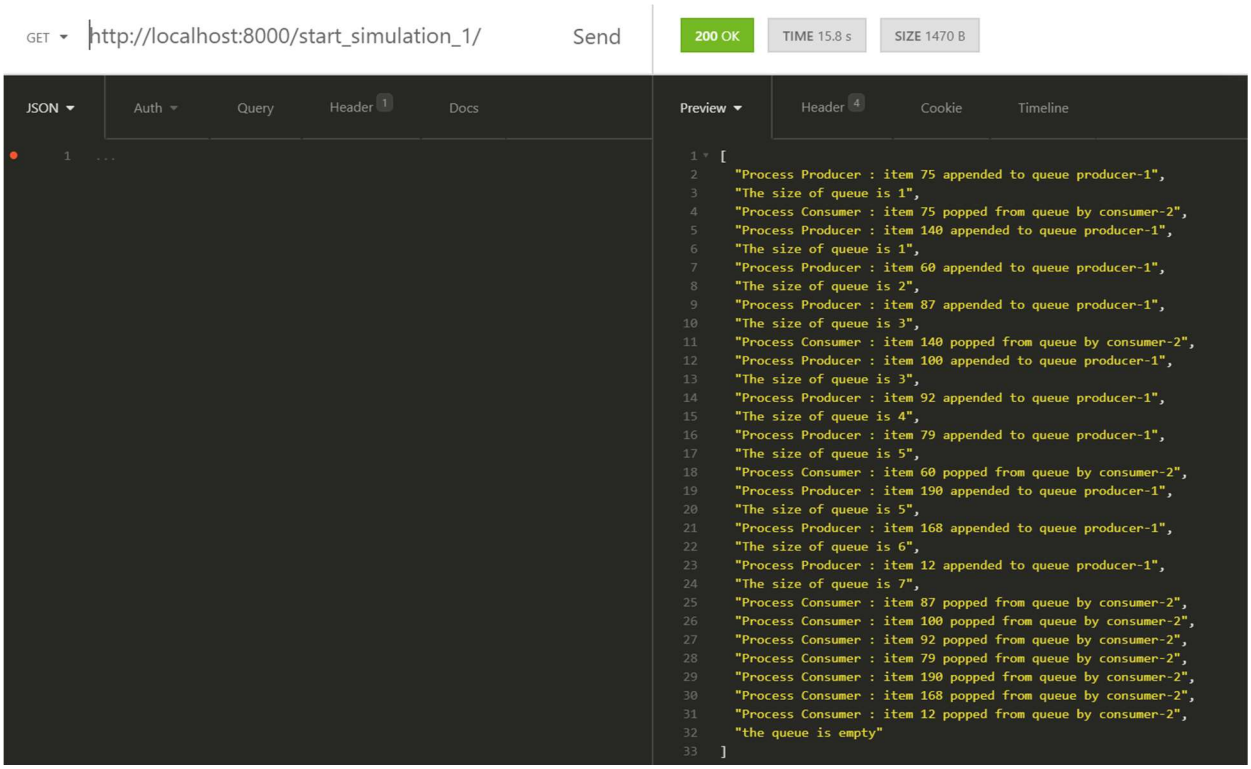
تابع consumer یک مصرف‌کننده را نمایش می‌دهد که به طور بی‌پایان اجرا می‌شود. اگر صف queue خالی نباشد، یک آیتم از آن خارج می‌شود و پیامی در log_queue قرار می‌دهد که نشان دهنده خروج آیتم از صف توسط مصرف‌کننده است. سپس با استفاده از time.sleep(1.5) تاخیری ایجاد می‌کند. اگر صف queue خالی باشد، به حالت انتظار می رود.

```python
@app.get("/start_simulation_1")
async def start_simulation():
    queue = Queue()
    log_queue = Queue()
    producers = [Process(target=producer, args=(queue, log_queue, 1))]
    consumers = [Process(target=consumer, args=(queue, log_queue, 2))]

    for p in producers:
        p.start()
    for c in consumers:
        c.start()
    for p in producers:
        p.join()

    # Wait for the queue to empty
    while not queue.empty():
        time.sleep(0.2)

    for c in consumers:
        c.terminate()
        c.join()

    logs = []
    while not log_queue.empty():
        logs.append(log_queue.get())

    logs.append("the queue is empty")
    return logs
```

دو صف `Queue` برای ارتباط بین تولیدکنندگان و مصرف‌کنندگان و برای ثبت وضعیت‌ها (log) ایجاد می‌شود.

در خطوط 991 و 992 لیست‌هایی از فرآیندها که هر کدام یک تولیدکننده یا یک مصرف‌کننده را به عنوان هدف دارند، ایجاد می‌شوند. تابع `producer` برای `producer_id=1` فراخوانی می‌شود و تابع `consumer` برای `consumer_id=2` فراخوانی می‌شود.

http://localhost:7000/start_simulation_1/



سناریو دوم:

```python
@app.get("/start_simulation_2")
async def start_simulation():
    queue = Queue()
    log_queue = Queue()
    producers = [Process(target=producer, args=(queue, log_queue, 1))]
    consumers = [Process(target=consumer, args=(queue, log_queue, 2))]

    for p in producers:
        p.start()
    for p in producers:
        p.join()
    for c in consumers:
        c.start()

    # Wait for the queue to empty
    while not queue.empty():
        time.sleep(0.2)

    for c in consumers:
        c.terminate()
        c.join()

    logs = []
    while not log_queue.empty():
        logs.append(log_queue.get())

    logs.append("the queue is empty")
    return logs
```

GET http://localhost:8000/start_simulation_2/ Send

200 OK | TIME 22.3 s | SIZE 1481 B

JSON | Auth | Query | Header 1 | Docs

```
1 ...
```

Preview | Header 4 | Cookie | Timeline

```
[
  "Process Producer : item 119 appended to queue producer-1",
  "The size of queue is 1",
  "Process Producer : item 192 appended to queue producer-1",
  "The size of queue is 2",
  "Process Producer : item 241 appended to queue producer-1",
  "The size of queue is 3",
  "Process Producer : item 210 appended to queue producer-1",
  "The size of queue is 4",
  "Process Producer : item 196 appended to queue producer-1",
  "The size of queue is 5",
  "Process Producer : item 194 appended to queue producer-1",
  "The size of queue is 6",
  "Process Producer : item 32 appended to queue producer-1",
  "The size of queue is 7",
  "Process Producer : item 156 appended to queue producer-1",
  "The size of queue is 8",
  "Process Producer : item 162 appended to queue producer-1",
  "The size of queue is 9",
  "Process Producer : item 140 appended to queue producer-1",
  "The size of queue is 10",
  "Process Consumer : item 119 popped from queue by consumer-2",
  "Process Consumer : item 192 popped from queue by consumer-2",
  "Process Consumer : item 241 popped from queue by consumer-2",
  "Process Consumer : item 210 popped from queue by consumer-2",
  "Process Consumer : item 196 popped from queue by consumer-2",
  "Process Consumer : item 194 popped from queue by consumer-2",
  "Process Consumer : item 32 popped from queue by consumer-2",
  "Process Consumer : item 156 popped from queue by consumer-2",
  "Process Consumer : item 162 popped from queue by consumer-2",
  "Process Consumer : item 140 popped from queue by consumer-2",
  "the queue is empty"
]
```

سناریو سوم:

```python
1 usage  new *
def producer6_3(queue, log_queue, producer_id):
    for _ in range(10):
        item = random.randint(1, 300)
        queue.put((item, producer_id))
        log_queue.put(f"Process Producer : item {item} appended to queue producer-{producer_id}")
        log_queue.put(f"The size of queue is {queue.qsize()}")
        time.sleep(1)

1 usage  new *
def consumer6_3(queue, log_queue, consumer_id):
    while True:
        if not queue.empty():
            item, producer_id = queue.get()
            log_queue.put(f"Process Consumer : item {item} popped from queue by consumer-{consumer_id}")
            time.sleep(1)
        else:
            time.sleep(0.1)  # Short sleep to avoid busy-waiting
```

```python
@app.get("/start_simulation_3")
async def start_simulation():
    queue = Queue()
    log_queue = Queue()
    producers = [Process(target=producer6_3, args=(queue, log_queue, 1))]
    consumers = [Process(target=consumer6_3, args=(queue, log_queue, 2))]

    for p in producers:
        p.start()
    for c in consumers:
        c.start()
    for p in producers:
        p.join()

    # Wait for the queue to empty
    while not queue.empty():
        time.sleep(0.2)

    for c in consumers:
        c.terminate()
        c.join()

    logs = []
    while not log_queue.empty():
        logs.append(log_queue.get())

    logs.append("the queue is empty")
    return logs
```

GET http://localhost:8000/start_simulation_3/ Send

200 OK TIME 12.3 s SIZE 1472 B

JSON Auth Query Header 1 Docs

Preview Header 4 Cookie Timeline

```
[
  "Process Producer : item 236 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 236 popped from queue by consumer-2",
  "Process Producer : item 94 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 94 popped from queue by consumer-2",
  "Process Producer : item 49 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 49 popped from queue by consumer-2",
  "Process Producer : item 29 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 29 popped from queue by consumer-2",
  "Process Producer : item 152 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 152 popped from queue by consumer-2",
  "Process Producer : item 39 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 39 popped from queue by consumer-2",
  "Process Producer : item 245 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 245 popped from queue by consumer-2",
  "Process Producer : item 174 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 174 popped from queue by consumer-2",
  "Process Producer : item 167 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 167 popped from queue by consumer-2",
  "Process Producer : item 64 appended to queue producer-1",
  "The size of queue is 1",
  "Process Consumer : item 64 popped from queue by consumer-2",
  "the queue is empty"
]
```

7) Synchronizing processes

سناریو اول:

```python
from multiprocessing import Process, Barrier, Queue
import datetime

2 usages  new *
def test_with_barrier(barrier, process_name, log_queue):
    time.sleep(2)  # simulate some work
    barrier.wait()  # wait for other processes to reach this point
    time_now = datetime.datetime.now()
    log_queue.put(f"{process_name} - test_with_barrier ----> {time_now}")

2 usages  new *
def test_without_barrier(process_name, log_queue):
    time.sleep(1)  # simulate some work
    time_now = datetime.datetime.now()
    log_queue.put(f"{process_name} - test_without_barrier ----> {time_now}")
```

تابع test_with_barrier یک `barrier` و نام پردازه و صف برای ثبت وقایع (`log_queue`) را به عنوان ورودی می‌گیرد.

با فراخوانی `time.sleep(2)` یک تأخیر دو ثانیه ایجاد می‌کند تا یک شبیه‌سازی از کار انجام دهد. سپس با `barrier.wait()` منتظر می‌ماند تا تمام پردازه‌های دیگر به این نقطه برسند. در نهایت، زمان فعلی را با `datetime.datetime.now()` دریافت کرده و متن مربوط به وقایع را به `log_queue` اضافه می‌کند.

تابع test_without_barrier نام پردازه (`process_name`) و صف برای ثبت وقایع (`log_queue`) را به عنوان ورودی می‌گیرد. با فراخوانی `time.sleep(1)` یک تأخیر یک ثانیه ایجاد می‌کند تا یک شبیه‌سازی از کار انجام دهد. سپس، زمان فعلی را با `datetime.datetime.now()` دریافت کرده و متن مربوط به وقایع را به `log_queue` اضافه می‌کند.

```python
@app.get("/barrier_processes_1/")
async def start_processes():
    barrier = Barrier(2)  # Barrier for two processes
    log_queue = Queue()
    processes = []

    p1 = Process(target=test_without_barrier, args=('process p1', log_queue))
    p2 = Process(target=test_without_barrier, args=('process p2', log_queue))
    processes.append(p1)
    processes.append(p2)

    p3 = Process(target=test_with_barrier, args=(barrier, 'process p3', log_queue))
    p4 = Process(target=test_with_barrier, args=(barrier, 'process p4', log_queue))
    processes.append(p3)
    processes.append(p4)

    for p in processes:
        p.start()
    for p in processes:
        p.join()

    logs = []
    while not log_queue.empty():
        logs.append(log_queue.get())

    return logs
```

در این روت، یک `barrier` با ظرفیت دو تعریف می‌شود. یک صف برای ثبت وقایع (`log_queue`) نیز ایجاد می‌شود. یک لیست از پردازه‌ها (`processes`) ایجاد می‌شود که شامل چهار پردازه است.

دو پردازه `p1` و `p2` بدون استفاده از `barrier` ایجاد می‌شوند و دو پردازه دیگر `p3` و `p4` با استفاده از `barrier` ایجاد میشوند.

http://localhost:7000/barrier_processes_1/

GET http://localhost:8000/barrier_processes_1/ Send

200 OK TIME 3.77 s SIZE 271 B

JSON Auth Query Header 1 Docs

Preview Header 4 Cookie Timeline

```
1 ...
```

```
1 [
2   "process p1 - test_without_barrier ----> 2024-07-04 02:51:36.431992",
3   "process p2 - test_without_barrier ----> 2024-07-04 02:51:36.479895",
4   "process p4 - test_with_barrier ----> 2024-07-04 02:51:37.469089",
5   "process p3 - test_with_barrier ----> 2024-07-04 02:51:37.469089"
6 ]
```

سناریو دوم:

```python
@app.get("/barrier_processes_2/")
async def start_processes():
    barrier = Barrier(2)  # Barrier for two processes
    log_queue = Queue()
    processes = []

    p1 = Process(target=test_without_barrier, args=('process p1', log_queue))
    processes.append(p1)
    p3 = Process(target=test_with_barrier, args=(barrier, 'process p3', log_queue))
    processes.append(p3)
    p2 = Process(target=test_without_barrier, args=('process p2', log_queue))
    processes.append(p2)
    p4 = Process(target=test_with_barrier, args=(barrier, 'process p4', log_queue))
    processes.append(p4)

    for p in processes:
        p.start()
    for p in processes:
        p.join()

    logs = []
    while not log_queue.empty():
        logs.append(log_queue.get())

    return logs
```

GET http://localhost:8000/barrier_processes_2/ Send

200 OK TIME 4.24 s SIZE 271 B

JSON Auth Query Header 1 Docs

Preview Header 4 Cookie Timeline

```
1 ...
```

```
1 [
2   "process p1 - test_without_barrier ----> 2024-07-04 02:56:02.145631",
3   "process p2 - test_without_barrier ----> 2024-07-04 02:56:02.209732",
4   "process p3 - test_with_barrier ----> 2024-07-04 02:56:03.196783",
5   "process p4 - test_with_barrier ----> 2024-07-04 02:56:03.196783"
6 ]
```

سناریو سوم:

```python
def test_with_barrier_3(barrier, process_name, log_queue):
    time.sleep(1)  # simulate some work
    barrier.wait()  # wait for other processes to reach this point
    time_now = datetime.datetime.now()
    log_queue.put(f"{process_name} - test_with_barrier ----> {time_now}")

def test_without_barrier_3(process_name, log_queue):
    time.sleep(2)  # simulate some work
    time_now = datetime.datetime.now()
    log_queue.put(f"{process_name} - test_without_barrier ----> {time_now}")
```

مقدار تاخیرها را تغییر دادیم.

```python
@app.get("/barrier_processes_3/")
async def start_processes():
    barrier = Barrier(2)  # Barrier for two processes
    log_queue = Queue()
    processes = []

    p1 = Process(target=test_without_barrier_3, args=('process p1', log_queue))
    p2 = Process(target=test_without_barrier_3, args=('process p2', log_queue))
    processes.append(p1)
    processes.append(p2)

    p3 = Process(target=test_with_barrier_3, args=(barrier, 'process p3', log_queue))
    p4 = Process(target=test_with_barrier_3, args=(barrier, 'process p4', log_queue))
    processes.append(p3)
    processes.append(p4)

    for p in processes:
        p.start()
    for p in processes:
        p.join()

    logs = []
    while not log_queue.empty():
        logs.append(log_queue.get())

    return logs
```

GET http://localhost:8000/barrier_processes_3/ Send

200 OK | TIME 3.97 s | SIZE 271 B

JSON | Auth | Query | Header 1 | Docs

Preview | Header 4 | Cookie | Timeline

```
[
  "process p4 - test_with_barrier ----> 2024-07-04 03:01:14.939147",
  "process p3 - test_with_barrier ----> 2024-07-04 03:01:14.939147",
  "process p1 - test_without_barrier ----> 2024-07-04 03:01:15.844235",
  "process p2 - test_without_barrier ----> 2024-07-04 03:01:15.892173"
]
```

8)Using a process pool

سناریو اول:

```
1206  from multiprocessing import Pool
1207  import json
1208
      1 usage  new *
1209  def square(number):
1210      return number * number
      new *
1211  @app.get("/process_pool_1/")
1212  async def calculate_squares(max_number: int = Query(default=100, ge=0)):
1213      with Pool(processes=4) as pool:  # Use a pool of 4 processes
1214          result = pool.map(square, range(max_number))
1215      response_dict = {"Pool": result}
1216      return json.dumps(response_dict)
```

تابع square یک عدد را به عنوان ورودی می‌گیرد و مربع آن را محاسبه می‌کند.

در این روت، یک پارامتر با نام `max_number` به عنوان ورودی در نظر گرفته می‌شود که حداکثر عددی است که قرار است مربع آنها محاسبه شود.

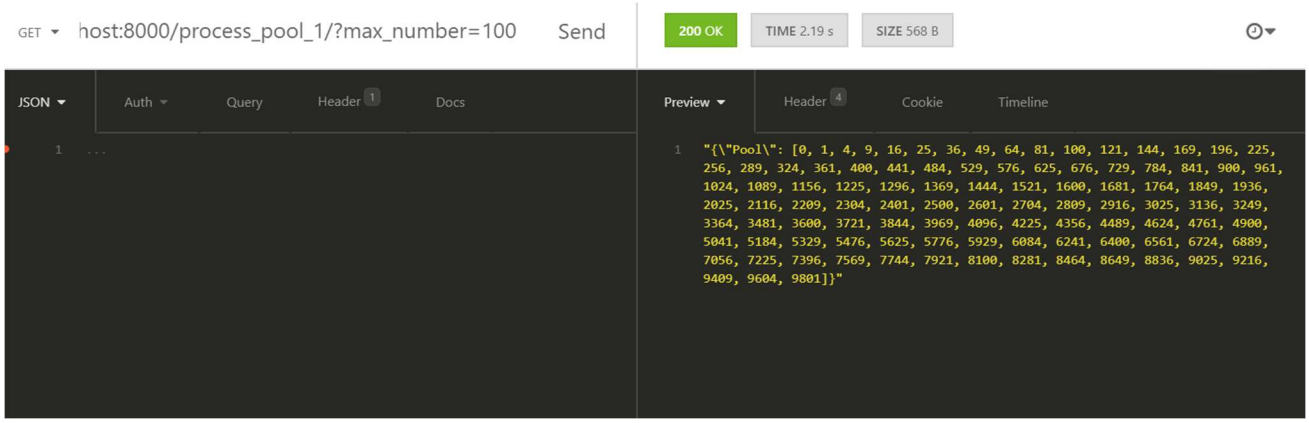
با استفاده از `Query(default=100, ge=0)`، پیش‌فرض برای `max_number` را ۱۰۰ تعیین می‌کنیم و همچنین مشخص می‌کنیم که `max_number` باید عددی نامنفی باشد.

خط 1213 یک استخر پردازنده با 4 پردازنده (processes=4) ایجاد می‌کند. استخر پردازنده به کاربر اجازه می‌دهد تا عملیات‌های مختلف را همزمان انجام دهد. با استفاده از `with`، استخر به طور خودکار می‌بندد و پس از اتمام کار منابع را آزاد می‌کند.

در اینجا، تابع `square` روی هر عدد در محدوده `range(max_number)` اعمال می‌شود و نتایج به صورت یک لیست در `result` ذخیره می‌شود.

در نهایت، این دیکشنری به صورت یک JSON string به عنوان خروجی به کلاینت برگردانده می‌شود.

http://localhost:7000/process_pool_1/?max_number=100

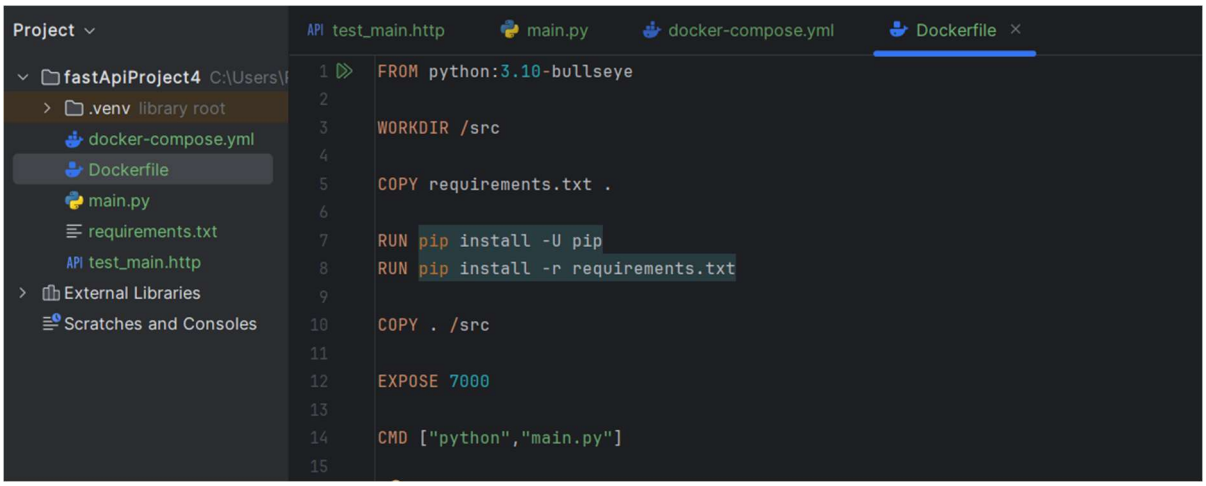


داکرایز

در فایل پروژه فایل های Dockerfile و docker-compose.yml را ایجاد میکنیم.

در فایل Dockerfile مسیری که پروژه باید طی کند را مینویسیم.

در فایل docker-compose.yml کانتینرها و image هایمان را مینویسیم.



در خط 1 گفتیم که برنامه برای اجرا به پایتون و ورژن مورد نظر نیاز دارد.

در خط 3 مسیری را تعیین کردیم که فایل پروژه در آن قرار گیرد.

در خط 5 فایل requirements.txt که در آن کتابخانه های لازم قرار دارند، در همان مسیر کپی میشود.

در خط 7 کتابخانه pip که برای نصب کتابخانه های پایتونی مورد نیاز است نصب میشود.

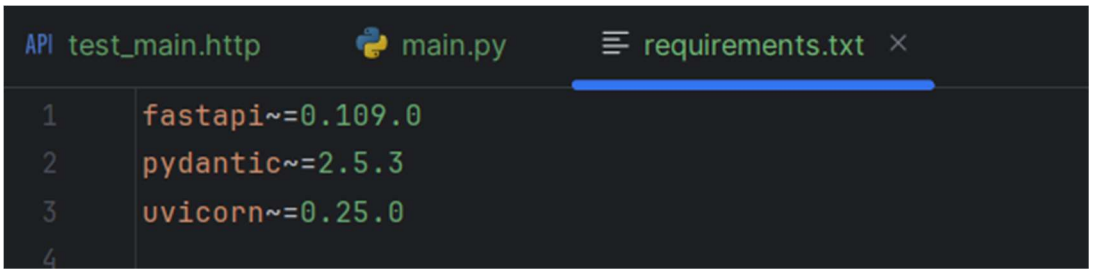
در خط 8 کتابخانه های داخل فایل requirements.txt را نصب میشوند.

در خط 10 همه ی فایل هایی که در فایل پروژه وجود دارند، در آن مسیری که تعیین کردیم کپی میشوند.

در خط 12 پورتی برنامه روی آن اجرا میشود گفته شده است.

در خط 14 دستوری که برنامه با آن اجرا میشود نوشته شده است.

فایل requirements.txt :



```
1  version: "3"
2
3  services:
4    app:
5      build: .
6      container_name: parallelproject
7      command: uvicorn --host 0.0.0.0 --port 7000 main:app --reload
8      ports:
9        - "7000:7000"
10     networks:
11       - main
12     restart: always
13
14 networks:
15   main:
16     driver: bridge
17
```

در خط 4 image پروژه را نوشتیم که app است.

در app image در خط 5 گفتیم که در همان مسیر فعلی ساخته شود و بعد برای این container نامی تعیین کردیم. خط 7 برای طریقه اجرا شدن پروژه است که میگوید با استفاده از uvicorn از هر هاستی و پورت 7000 و از فایل main برنامه را اجرا کن. در خط 8 پورت 7000 سیستم را به پورت 7000 این app وصل میکند. .در خطوط بعدی نام network مورد نظر را نوشتیم و در اخر در خط 12 تعیین کردیم که container در صورت به وجود آمدن مشکل خودش همیشه restart شود.

در نهایت در خط 14، network را تشکیل میدهیم.

برای ساخت image پروژه وارد پوشه پروژه میشویم و با دستور زیر image را میسازیم:

```
C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4>docker-compose up -d
[+] Building 39.1s (11/11) FINISHED                                                    docker:default
 => [app internal] load build definition from Dockerfile                                          0.1s
 => => transferring dockerfile: 234B                                                              0.0s
 => [app internal] load .dockerignore                                                             0.1s
 => => transferring context: 2B                                                                   0.0s
 => [app internal] load metadata for docker.io/library/python:3.10-bullseye                       0.0s
 => [app 1/6] FROM docker.io/library/python:3.10-bullseye                                         0.0s
 => [app internal] load build context                                                             3.2s
 => => transferring context: 49.68MB                                                              3.2s
 => CACHED [app 2/6] WORKDIR /src                                                                 0.0s
 => [app 3/6] COPY requirements.txt .                                                             0.5s
 => [app 4/6] RUN pip install -U pip                                                             13.4s
 => [app 5/6] RUN pip install -r requirements.txt                                                18.6s
 => [app 6/6] COPY . /src                                                                         2.3s
 => [app] exporting to image                                                                      0.9s
 => => exporting layers                                                                           0.9s
 => => writing image sha256:064c25f136285ea03e807db8ad4b58c4b7a8064aa79b91a06dad150dfa979494      0.0s
 => => naming to docker.io/library/fastapiproject4-app                                            0.0s
[+] Running 2/2
 ✔ Network fastapiproject4_main  Created                                                          0.1s
 ✔ Container parallelproject     Started                                                          0.2s
```

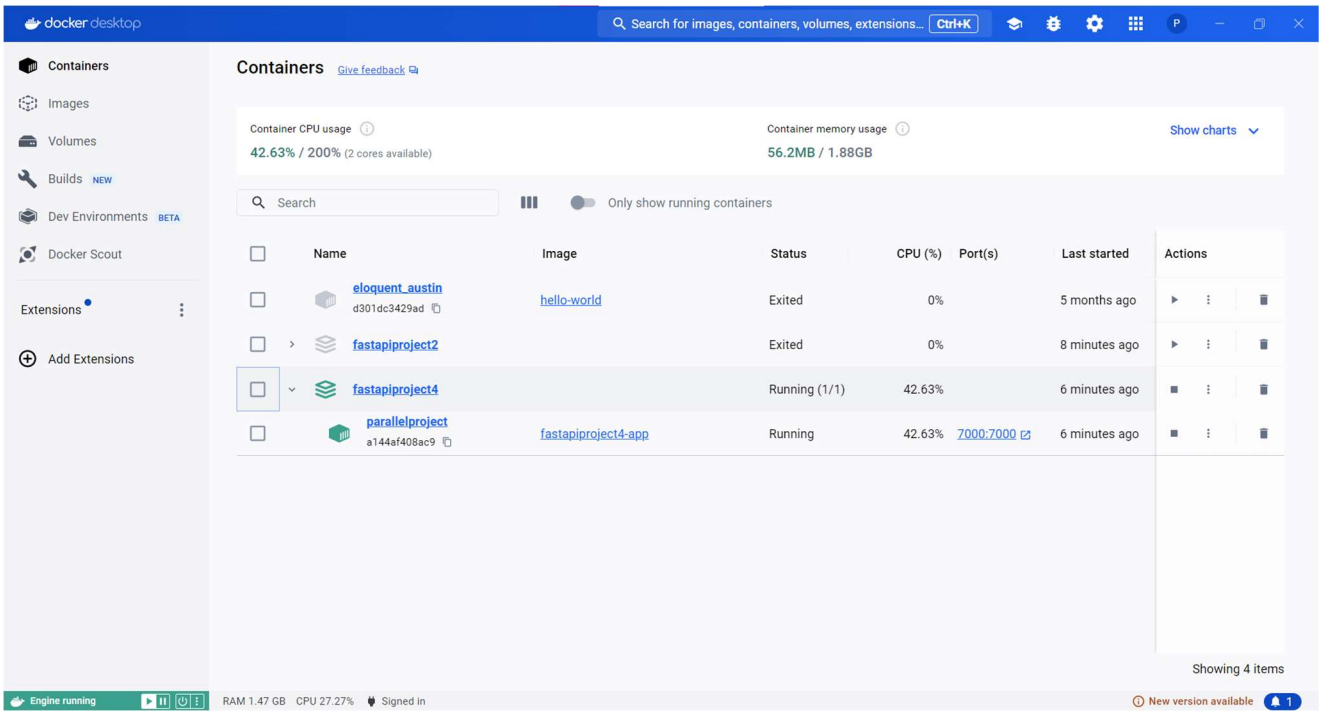
```
C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4>docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS         PORTS                     NAMES
a144af408ac9   fastapiproject4-app  "uvicorn --host 0.0.…"   2 minutes ago   Up 2 minutes   0.0.0.0:7000->7000/tcp    parallelproject
2beb0bba70f3   fastapiproject2-app  "uvicorn --host 0.0.…"   5 months ago    Up 4 minutes   0.0.0.0:8000->8000/tcp    fastprojectmongo
2c83c7ceae21   mongo:6.0.13         "docker-entrypoint.s…"   5 months ago    Up 4 minutes   0.0.0.0:27017->27017/tcp  fastmongo

C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4>docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED         STATUS                     PORTS                     NAMES
a144af408ac9   fastapiproject4-app  "uvicorn --host 0.0.…"   3 minutes ago   Up 3 minutes               0.0.0.0:7000->7000/tcp    parallelproject
2beb0bba70f3   fastapiproject2-app  "uvicorn --host 0.0.…"   5 months ago    Up 4 minutes               0.0.0.0:8000->8000/tcp    fastprojectmongo
2c83c7ceae21   mongo:6.0.13         "docker-entrypoint.s…"   5 months ago    Up 4 minutes               0.0.0.0:27017->27017/tcp  fastmongo
d301dc3429ad   hello-world          "/hello"                 5 months ago    Exited (0) 5 months ago                              eloquent_austin

C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4>docker images
REPOSITORY           TAG            IMAGE ID       CREATED         SIZE
fastapiproject4-app  latest         064c25f13628   3 minutes ago   992MB
fastapiproject2-app  latest         19cb94ac15ab   5 months ago    999MB
fastapiproject3-app  latest         941ba8fa3200   5 months ago    1.02GB
mongo                6.0.13         23e54c868737   5 months ago    690MB
postgres             16.1           75b7bff7c3ad   6 months ago    425MB
python               3.10-bullseye  2efb40c32a8e   6 months ago    911MB
hello-world          latest         d2c94e258dcb   14 months ago   13.3kB
```

دستور docker ps لیست container های فعال را به ما نشان میدهد که همانطور که مشاهده میکنید contanner که ما ساختیم هم با نام های fastapipeoject4-app فعال است.

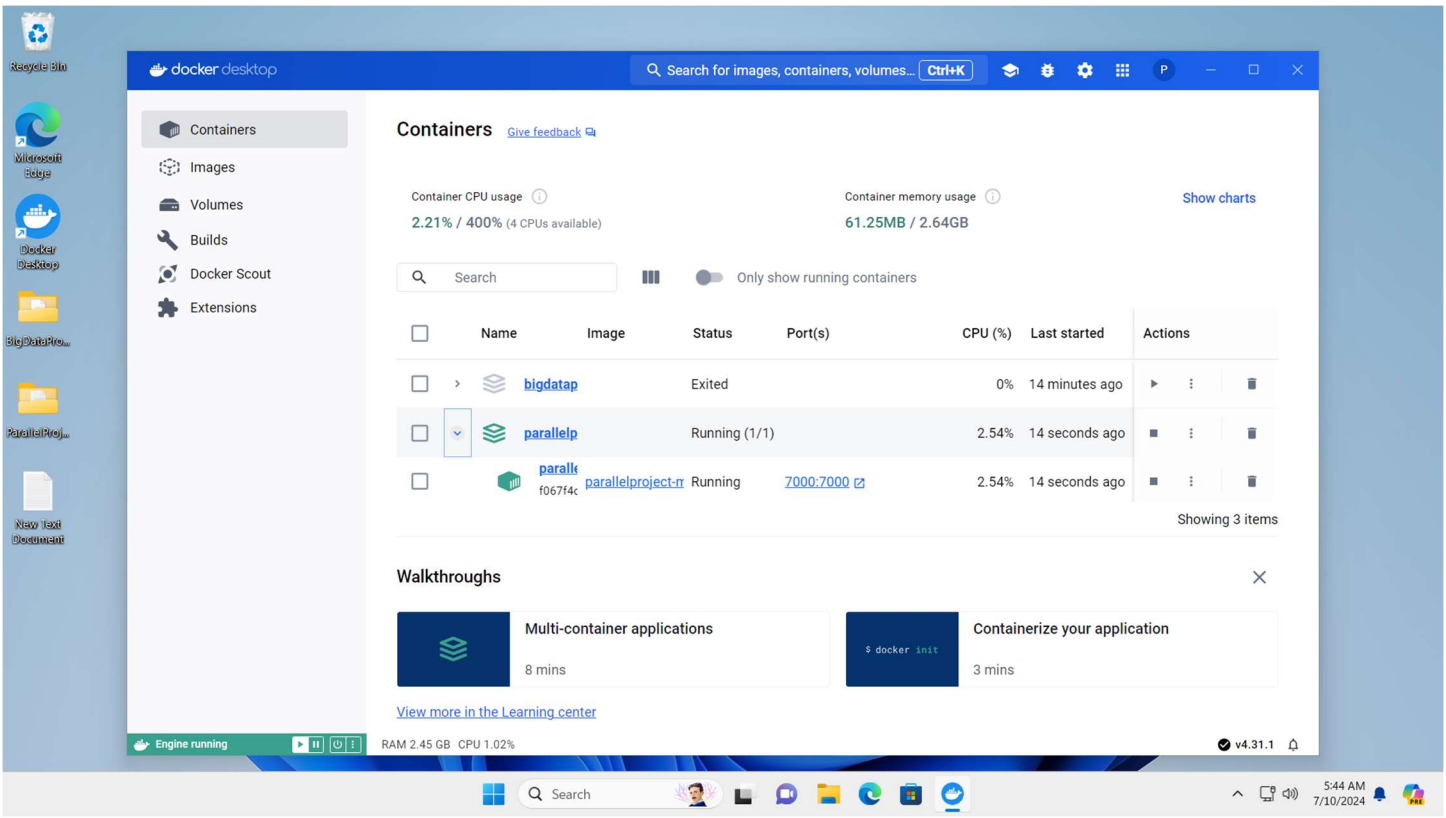
دستور docker ps –a لیست همه ی container ها چه فعال و چه غیر فعال را نشان میدهد.

و در اخر با دستور docker images لیست image ها را میبینیم که fastapipeoject4-app هم که مربوط به پروژه ما هست اضافه شده است.

همانطور که مشاهده میکنید در Docker هم container ساخته شده را میتوانیم ببینیم:



داکر در سرور


docker desktop
Search for images, containers, volumes... Ctrl+K
Containers
Images
Volumes
Builds
Docker Scout
Extensions
Containers
Give feedback
Container CPU usage
2.21% / 400% (4 CPUs available)
Container memory usage
61.25MB / 2.64GB
Show charts
Search
Only show running containers
Name
Image
Status
Port(s)
CPU (%)
Last started
Actions
bigdatap
Exited
0%
14 minutes ago
parallelp
Running (1/1)
2.54%
14 seconds ago
parallelproject-m
Running
7000:7000
2.54%
14 seconds ago
Showing 3 items
Walkthroughs
Multi-container applications
8 mins
Containerize your application
3 mins
View more in the Learning center
Engine running
RAM 2.45 GB CPU 1.02%
v4.31.1

چند نمونه خروجی در سرور

GET http://45.252.182.94:7000/barrier_1 Send

200 OK | TIME 19.8 s | SIZE 380 B

JSON | Auth | Query | Header 1 | Docs

```
1 ...
```

Preview | Header 4 | Cookie | Timeline


```
1  [
2    "START RACE!!!!",
3    "Dewey reached the barrier at: 2024-07-10 02:16:25.168228",
4    "Huey reached the barrier at: 2024-07-10 02:16:25.182689",
5    "Louie reached the barrier at: 2024-07-10 02:16:25.222237",
6    "Louie passed the barrier at: 2024-07-10 02:16:25.225926",
7    "Huey passed the barrier at: 2024-07-10 02:16:25.226661",
8    "Dewey passed the barrier at: 2024-07-10 02:16:25.227508",
9    "Race over!"
10 ]
```


GET http://45.252.182.94:7000/run_processes_2 Send

200 OK | TIME 18 s | SIZE 325 B

JSON | Auth | Query | Header 1 | Docs

```
1 ...
```

Preview | Header 4 | Cookie | Timeline


```
1  [
2    "Starting NO_background_process",
3    "---> 5",
4    "---> 6",
5    "---> 7",
6    "---> 8",
7    "---> 9",
8    "Exiting NO_background_process",
9    "Starting NO_background_process",
10   "---> 5",
11   "---> 6",
12   "---> 7",
13   "---> 8",
14   "---> 9",
15   "Exiting NO_background_process",
16   "Starting background_process",
17   "---> 0",
18   "---> 1",
19   "---> 2",
20   "---> 3",
21   "---> 4",
22   "Exiting background_process"
23 ]
```


GET http://45.252.182.94:7000/process_pool_1/?max_number=100 Send

200 OK | TIME 2.7 s | SIZE 568 B

JSON | Auth | Query | Header 1 | Docs

```
1 ...
```

Preview | Header 4 | Cookie | Timeline

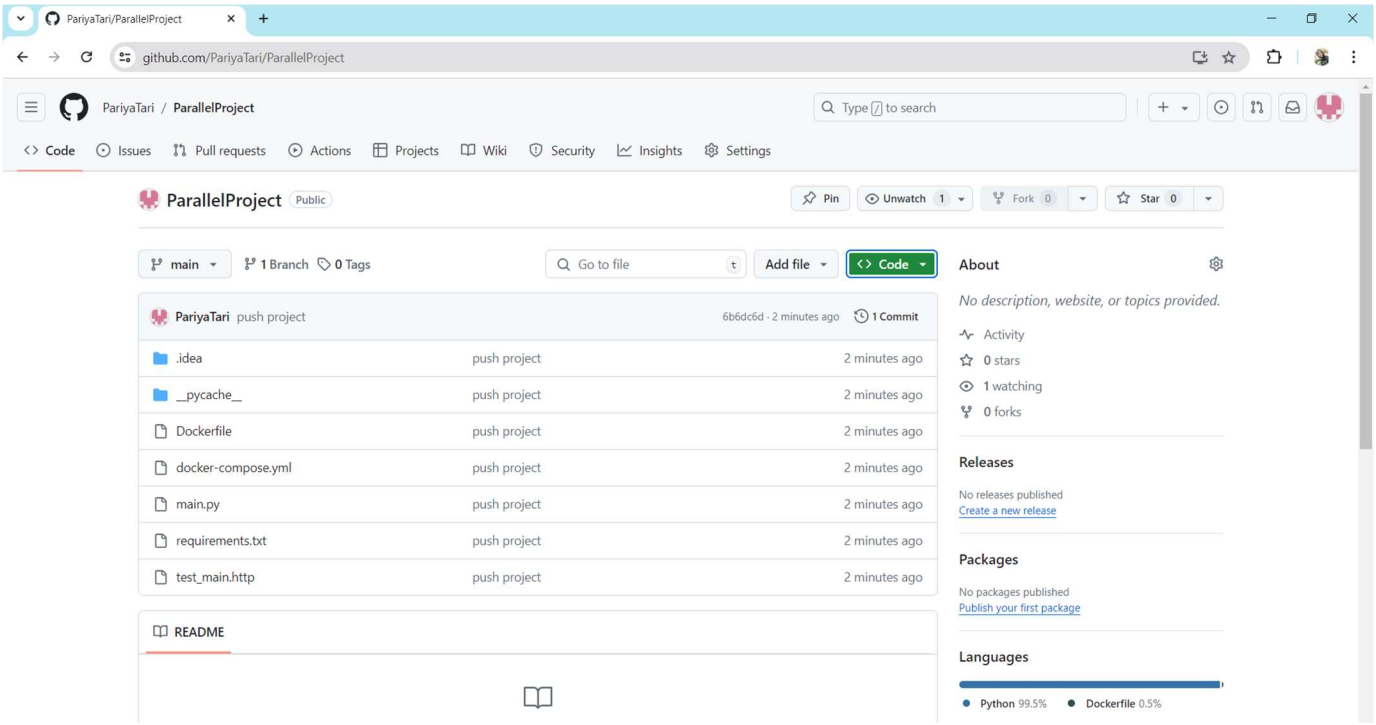

```
1  "{\"Pool\": [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100,
   121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441,
   484, 529, 576, 625, 676, 729, 784, 841, 900, 961,
   1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600,
   1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401,
   2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364,
   3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489,
   4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776,
   5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225,
   7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836,
   9025, 9216, 9409, 9604, 9801]}"
```

Github

```
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4> git init
Reinitialized existing Git repository in C:/Users/RatinRayaneh/PycharmProjects/fastApiProject4/.git/
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4> git remote add origin git@github.com:PariyaTari/ParallelProject.git
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4> git branch -M main
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4> git add .
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4> git commit -m "push project"
[main (root-commit) 6b6dc6d] push project
 12 files changed, 1310 insertions(+)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/fastApiProject4.iml
 create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 Dockerfile
 create mode 100644 __pycache__/main.cpython-39.pyc
 create mode 100644 docker-compose.yml
 create mode 100644 main.py
 create mode 100644 requirements.txt
 create mode 100644 test_main.http
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4> git push -u origin main
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 8 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (17/17), 16.00 KiB | 1.14 MiB/s, done.
Total 17 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:PariyaTari/ParallelProject.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
(.venv) PS C:\Users\RatinRayaneh\PycharmProjects\fastApiProject4>
```

عکس مراحل پوش کردن پروژه به repository در گیتهاب را نشان میدهد.

ثبت پروژه در github :



❊پایان❊